PHRED: A Generator for Natural Language Interfaces*

Paul S. Jacobs

Berkeley Artificial Intelligence Research
Division of Computer Science
Department of EECS
University of California
Berkeley, CA, USA

ABSTRACT

PHRED (PHRasal English Diction) is a natural language generator designed for use in a variety of domains. It was constructed to share a knowledge base with PHRAN (PHRasal ANalyzer) as part of a real-time user-friendly interface. The knowledge base consists of *pattern-concept pairs*, i.e., associations between linguistic structures and conceptual templates. Using this knowledge base, PHRED produces appropriate and grammatical natural language output from a conceptual representation.

PHRED and PHRAN are currently used as central components of the user interface to the UNIX Consultant System (UC). This system answers questions and solves problems related to the UNIX operating system. UC passes the conceptual form of its responses, usually either questions or answers to questions, to the PHRED generator, which expresses them in the user's language. Currently the consultant can answer questions and produce its responses in either English or Spanish.

There are a number of practical advantages to PHRED as the generation component of a natural language system. Having a knowledge base shared between analyzer and generator eliminates the inevitable redundancy of having separate grammars and lexicons for input and output. It avoids possibly awkward inconsistencies caused by such a separation, and allows for interchangeable interfaces, such as the English and Spanish versions of the UC interface.

The phrasal approach to language processing realized in PHRED has proven helpful in generation as in analysis. PHRED commands the use of idioms, grammatical constructions, and canned phrases without a specialized mechanism or data structure. It does so without restricting its ability to utilize more general linguistic knowledge.

While PHRED affords extensibility, simplicity, and processing speed, its design incorporates a cognitive motivation as well. It diverges from the traditional computational approach to language as a symbol manipulation process, and treats it more as an associative process among knowledge structures. The phrasal approach minimizes the autonomy of the individual word, the bane of some Artificial Intelligence approaches to language. The treatment of most linguistic knowledge as declarative bears cognitive as well as practical significance. The two-stage process used by PHRED to select appropriate linguistic structures also fits well with cognitive theories of language and memory.

PHRED: A Generator for Natural Language Interfaces*

Paul S. Jacobs

Berkeley Artificial Intelligence Research
Division of Computer Science
Department of EECS
University of California
Berkeley, CA, USA

1. Introduction

Computer programs which can effectively communicate in natural language must be capable both of analyzing a wide range of utterances to derive their meaning or intent, and of producing appropriate and intelligible responses. Historically these two tasks have been treated independently, principally because some of the hard problems in language production differ from those of language analysis. In the MARGIE system, for example, the BABEL generator (Goldman, 1975) employed a discrimination net as its principal data structure to facilitate the selection of an appropriate verb and an ATN grammar to apply syntactic constraints, while the ELI analyzer (Riesbeck, 1975) in the same system attached routines to individual words to control the interpretations considered during the parsing process.

Throughout the short history of natural language generation systems, programs which produce language have treated generation as a process of *decision making* (McDonald, 1980), *choice* (Mann and Matthiessen, 1983), or *planning* (Appelt, 1982). These systems have employed knowledge structures specifically geared, to varying degrees, to the task of constraining the selection of lexical and grammatical elements. The design of analyzers, on the other hand, focuses on the problem of ambiguity in natural language and makes use of knowledge structures designed to constrain the consideration of alternative interpretations.

Even in systems with both analysis and generation components, the knowledge used to derive meaning from language is not used to produce language from meaning. Such systems may be able to use a word or grammatical structure without being able to recognize the same structure, or *vice versa*, and must duplicate a great deal of information if the generator uses language similar to that understood by the analyzer. Intuitively, it seems that the knowledge used to constrain the interpretation of language can be used to constrain the choice of language. A cognitively realistic natural language system with a parsimonious knowledge representation could encompass an interface capable of both analysis and production without excessive duplication, with knowledge *about* language as well as mechanisms for its analysis and generation.

* This research was sponsored in part by the Office of Naval Research under contract N00014-80-C-0732, the National Science Foundation under grants IST-8007045 and IST-8208602, and the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3041, Monitored by the Naval Electronic Systems Command under contract N00039-82-C-0235. I am grateful to Robert Wilensky for his guidance and for his important comments on numerous drafts of this paper, and to Lisa Rau for many helpful suggestions.

PHRED (PHRasal English Diction) was conceived as the generation component of such an interface. PHRED, along with its companion analysis program PHRAN (PHRasal ANalyzer) (Wilensky and Arens, 1980), embodies an approach to natural language processing founded on the principle that knowledge about language is in essence declarative, consisting of associations between linguistic and conceptual structures of varying degrees of specificity (Wilensky, 1981). Such declarative knowledge should be used for both analysis and generation, and its knowledge structure independent of the particular language. From its initial conception the "English" part of PHRED's acronym was anomalous, as both the analyzer and generator were envisioned as being able to utilize knowledge bases of multiple languages.

PHRED and PHRAN currently make up the linguistic component of the user interface for the UNIX Consultant System (UC) (Wilensky, Arens, and Chin, 1984), a program designed to aide novice users of the UNIX operating system by conversing in natural language.. PHRAN analyzes the user's input to UC, which then passes its response in conceptual form to PHRED to be converted into natural language output. Below are some examples of questions to UC, with the responses generated by PHRED:

User: How do I delete a file?

UC: **Typing 'rm filename' causes the file filename to be removed from the current directory.**

User: Cómo se borra un archivo?

UC: **Para borrar el archivo de nombre filename del espacio de trabajo, se escribe 'rm filename'.**

User: How do I delete all the files from a directory?

UC: **Typing 'rm *' will remove all files within the current directory from the current directory.**

Typing 'rm -r directoryname' will cause all files and directories recursively inside of directoryname to be deleted from directoryname and removes the directory directoryname from the working directory.

While PHRAN and PHRED were originally tested using an English vocabulary used for various stories and news articles, it was a relatively easy task to accommodate linguistic knowledge bases for English and Spanish in order for the same programs to operate in the UC domain. Adding a new vocabulary or language capability to the UC system has required no modification to the program, although the system has not had extensive testing with many languages.

PHRED is implemented in Franz LISP and runs compiled on a VAX 11/780. The English linguistic knowledge base of UC contains about 150 patterns, in addition to knowledge of the morphological characteristics of 30 verbs and 50 nouns. The compiled program occupies about 100K bytes of memory, of which about 20K is code used also by PHRAN. Output from PHRED in the UC system requires 1-3 seconds of CPU time, roughly a third of the total time used by the system.

2. Design

The design and construction of the PHRED generator is the product of three interacting goals:

(1) *robustness*

-- A useful language generator must produce a variety of correct utterances, with potentially complex and varied syntax, without being limited to a particular specialized domain.

(2) *cognitive validity*

-- There is a growing body of evidence which sheds light on the way people use language and access linguistic information. This includes data concerning linguistic and conceptual categories and prototypes, associative priming, language acquisition, use of speech acts, and conversational tendencies. A system which takes this evidence into account is more likely to be extensible and integrable with other cognitive mechanisms.

(3) *efficiency*

-- Efficiency of knowledge representation plays a role in determining not only the space usage and search time of a system, but also the ease with which new knowledge can be added. There are thus three measures of efficiency to be considered: *size*, the amount of space used by the knowledge of a system, *access*, the speed with which knowledge can be retrieved, and *extensibility*, the amount of space required to add new knowledge to the system. The three measures are correlated, but there are often tradeoffs among them in the development of a natural language system.

2.1. Representation

PHRED, like PHRAN, employs a knowledge base consisting of pattern-concept pairs, which attach a conceptual structure or template to a linguistic pattern or expression. The use of the pattern-concept pair as the unit of linguistic knowledge representation, principally motivated by the desire to have a database shared by analyzer and generator, has brought PHRED closer to achieving the above three goals. For a discussion of the details of this representation and how it relates in particular to language analysis, the reader is referred to Wilensky and Arens (1980).

The underlying scheme for linguistic representation in PHRED is the same as that originally proposed for PHRAN. While the actual knowledge structures were modified to give PHRED a more powerful mechanism for handling agreement among constituents and for determining the order in which constituents are generated, the principal design of the PHRAN/PHRED representation is the same as in Wilensky's description (Wilensky, 1981). This design has the following essential features:

(1) *declarative nature*

The pattern-concept pairs used in PHRAN/PHRED are associations between linguistic structures and their conceptual and functional representations. Historically, the declarative representation was motivated in part by problems with analyzers such as Riesbeck's (Riesbeck, 1975) which relied on procedural knowledge. Declarative theories of linguistic representation have since become more prevalent; however, some of the knowledge schemes used in generation, such as systemic grammar (Halliday, 1968) and Incremental Procedural Grammar (Kempen and Hoenkamp, 1982) have intimate procedural ties.

(2) *phrasal patterns*

The pattern component of the PHRED knowledge structures may represent general syntactic information or knowledge about a particular idiom. Much of linguistic knowledge is based not around individual words, nor around general syntactic entities, but rather around phrases of varying degrees of flexibility. Many systems employ "phrasal lexicons" to handle "canned" or fixed phrases which behave in particular ways, but either ignore or handle as special cases the unfixed idioms and other grammatical constructions not present in "core" theories of grammar. The phrasal patterns in PHRED facilitate the use of such elements without treating them as special cases.

(3) *interaction among features*

The PHRAN/PHRED framework allows for a grammatical constituent to be specified using a combination of syntactic, functional, and semantic features. The syntactic category of a pattern, treated as a special property in many systems, is an undistinguished attribute to a pattern-concept pair. While the pattern-concept pairs used by PHRED are similar to the knowledge structures in unification grammar (Kay, 1983), PHRED differs from most generators which employ a unification grammar in the facilitation of the interaction of syntax with other types of knowledge.

The pattern-concept pair representation has developed is parallel with research on formulaicity and idiomaticity done primarily by cognitive linguists. Much of this work questions the cognitive validity of traditional generative theories of grammar. Chafe (1968) identifies certain idioms, such as "by and large" and "all of a sudden" which would be ungrammatical were they not given special status as idiomatic constructions. Other expressions, such as "kick the bucket", are grammatical, but have a meaning which is not determined by any compositional relationship among their components. Chafe argues that these idiomatic constructs sufficiently pervade everyday language to warrant an approach to language which handles these constructs not as special cases or exceptions but as an integral part of a language.

Becker (1975) presents the idea of the phrasal lexicon as a means of handling canned and idiomatic phrases. Becker identifies in particular a range of phrases which are grammatical and even comprehensible via compositional rules, yet which suggest specialized contextual knowledge. The expression "It only hurts when I laugh" can theoretically be handled using traditional theories of grammar, but treating it as such would be ignoring an important component of the expression's meaning. The existence of such expressions, which involve either

partially or entirely specialized knowledge, has generally been treated as of minor importance in computational theories of language. However, a cognitively realistic representation must take into account the role of both general syntactic knowledge and specialized knowledge about particular phrases.

While these arguments are directed at developing cognitively valid theories of linguistic representation, the handling of idiomaticity and of specialized phrasal knowledge has a substantial influence on the robustness and efficiency of a system. If specialized linguistic knowledge is indeed as pervasive as Chafe would argue, a system which deals only with "core" grammatical and productive constructs will handle but a small portion of a language. A generator working within such a system would be severely limited in the range of utterances which it could produce and in its ability to produce an output appropriate to a given context. On the other hand, failing to take advantage of linguistic generalizations can introduce redundancy and possibly inefficiency into the knowledge base. Robust and efficient language processing therefore demands a representation which takes advantage of both specialized idiomatic and general syntactic knowledge.

Fillmore (1979) gives arguments for the idea of the *structural formula*, a phrase or construction which cannot be described strictly as the composition of its components but may still have a certain degree of structural freedom. Fillmore presents "<Time unit> in and <Time unit> out" as an example of such a formula, manifest in expressions such as "day in and day out" and "week in and week out". More recently, Fillmore and others extend this idea to a theory of *grammatical constructions* (cf. Fillmore, Kay, and O' Connor, 1984; Lakoff, 1984), focusing on expressions which exhibit certain regularities and obey some grammatical constraints but whose behavior cannot be determined by "core" grammar. Examples of such expressions are "let alone" as in "He didn't make first lieutenant, let alone general", and the deictic "there", as in "There goes Harry, shooting his mouth off again". Fillmore, Kay, and O' Connor point out the difference between attempting to develop a minimal base of knowledge from which a linguistic competence can be *computed*, and attempting to develop a knowledge base which represents how human linguistic knowledge is in fact *stored*.

As an example of this distinction, consider the division drawn by Fillmore, Kay, and O' Connor between idioms of *decoding*, such as "kick the bucket", and "spill the beans", and idioms of *encoding* only, such as "answer the door", and "wide awake." All of these are grammatical idioms; that is, they have a syntactic structure and word order compatible with core grammatical constructs. The idioms of decoding, however, require specialized knowledge both for the comprehension of their meaning and their appropriate use. The idioms of encoding could possibly be comprehended using knowledge about their components only, but specialized knowledge is required to predict their use. Whether this specialized knowledge is to be stored in a given representational model therefore depends on what problem the model is addressing: competence, comprehension, or production.

We have thus distinguished three potential classes of linguistic knowledge: (1) The knowledge which is required to determine the *membership* of a given phrase or sentence in a language, (2) that which is necessary to determine the *meaning* of a phrase, and (3) that which determines appropriate *use* of the phrase. Modern computational linguistics has emphasized the first class, and thus many systems have attempted to define the second and third knowledge classes by adding auxiliary knowledge to a grammar for a linguistic competence. The PHRAN/PHRED pattern-concept pair representation, on the other hand, attempts to *subsume* the three classes into a single framework. Since the goal of PHRAN and PHRED is

proficient analysis and use of language, the distinction between grammatical and extragrammatical idioms becomes of minor importance. It is counterintuitive to treat phrases such as "all of a sudden" as of a different nature from "kick the bucket" simply because the former is extragrammatical. Further, the emphasis on the ability to compute a linguistic competence using a small set of rules is diminished. If specialized knowledge about a given phrase is required for its appropriate use, there is no reason why this knowledge cannot also be used for its syntactic analysis, even if in a system which performs analysis alone such knowledge would be redundant.

Consider the phrase "answer the door". A pure syntactic analyzer would require no special knowledge to recognize the construct as a valid verb phrase. It is possible as well that the meaning of the phrase could be determined based on the structure of the verb phrase and its constituents. However, in order for PHRED to give the phrase its deserved distinction from "respond to the door" or other less appropriate utterances, special knowledge, that "answer the door" means to open a door in response to a knock or doorbell, is required. Since this knowledge is encoded into the common knowledge base, it may also be used by PHRAN to determine the meaning of the phrase.

The development of a knowledge base for the purposes of both language analysis and language production therefore changes the nature of the linguistic knowledge base and its use. Information which is redundant when considered from a strict computational standpoint may be essential for a particular aspect of language processing. Such specialized knowledge may then be used by other components of the system. Thus the emphasis in the PHRAN/PHRED representation is on the *storage* of such redundant information rather than on its *computation*. Specialized knowledge about phrases and constructions is an integral part of the knowledge base and is used preferentially to general knowledge which requires more computation, both for analysis and production.

Of course, fundamental differences between analysis and generation still exist in PHRAN and PHRED. While the two programs have a shared knowledge base, they have entirely independent methods of accessing and applying their linguistic knowledge. PHRAN accesses patterns by recognizing sequences of constituents; PHRED must select a pattern based on the concept it is to express and the constraints which the pattern must satisfy. The PHRED approach to language generation is committed to the representation of linguistic knowledge in a declarative form which can be shared by the analyzer. The knowledge structures used by the generator are the same as those used by the analyzer, but the *process* which makes use of this knowledge to produce an utterance still reflects the basic choice problem.

2.2. The Generation Process

The process by which PHRED produces an utterance from its conceptual representation involves three stages. *Fetching* is the process of retrieving candidate patterns from the knowledge base of pattern-concept pairs. *Restriction* is the matching of a candidate pattern to the specifications derived from the concept and the addition of constraints to a selected pattern. *Interpretation* is the production of grammatical output from the chosen pattern.

The fetching mechanism of PHRED uses a multiple-key hashing mechanism to produce a stream of possible pattern-concept pairs, given a conceptual form and/or a set of constraints that the pattern must satisfy. In the restriction phase a possible pattern is evaluated to determine whether it in fact fits the concept and

does not violate any of the constraints. The basic choice problem therefore encompasses two different phenomena, which may be viewed theoretically as *predisposition* and *selection*. Predisposition is the process by which access to a knowledge base is influenced by various factors – such as the context, the concept to be expressed, or specific constraints on the desired output – to influence the order or priority in which elements of the knowledge base are considered. Selection is the evaluation of an element from the knowledge base. Intuitively, predisposition is the underlying access process which influences the likelihood of considering a particular word or phrase; selection is the judgement process which determines whether the word or phrase is appropriate.

There are three motivations for a design which provides for both a predisposition and a selection phase of the choice process. First, a system which employs as its principal choice mechanism, for example, a discrimination net such as Goldman's (Goldman, 1975) or a unification scheme such as McKeown's (McKeown, 1982) may apply its choice algorithm to many unlikely candidates, sometimes causing inefficiency. For example, the system might consider the verbs "smoke" and "inhale" every time it chooses the verb "breathe". A fast indexing mechanism which quickly selects candidates trims the time spent evaluating inappropriate choices.

The second motivating force lies in the distinction between utterances which are technically correct in expressing a given concept and those which are generally appropriate to a given context. "John inhaled air" is technically correct but generally inappropriate in place of "John breathed". This type of distinction can be embedded in a choice mechanism by attempting to axiomatize the rules which determine appropriateness, or it can be embedded in a predisposition mechanism which happens to order the choices according to the context. Predisposition thus provides a means for biasing choice without blurring the distinction between correctness and appropriateness.

The third motivation is cognitive validity. The predisposition-selection distinction fits the intuition that people have when they hear an unusual sentence: "It's okay but I wouldn't say it." In the example of "breathe" and "inhale air" both utterances may fit the input conceptualization, but fluent speakers tend to choose the former. Fluent speakers also bias their predisposition mechanisms according to the nature and formality of the context. Pawley and Syder (1980) find that one of the differences between native and non-native speakers of a language is that non-native speakers take a long time to develop the predisposition component necessary for fluency. Chafe has pointed out some of the influential factors in the variations between spoken and written, or informal and formal, language (Chafe, 1984). While some of this work is still in its early stages, the evidence strongly suggests a contextual biasing component distinct from the selection or evaluation phase of production.

The goal behind the PHRED indexing scheme is to incorporate as much of the choice problem as possible into the fetching, or predisposition, phase. Some language generators (Goldman, 1975; McDonald, 1980) use indexing tools which model choice as a multistage evaluation or decision-making process. The division of this process into an "automatic" biasing component and a judgment component has some practical advantages. The hashing algorithm which drives the fetching mechanism orders the stream of patterns retrieved before any of them is actually evaluated, and thus the more time-consuming restriction process is spared having to apply heuristics to make certain choices. For example, a general heuristic used by a number of language generators can be expressed as "Choose the most specific pattern which matches the input constraints". In PHRED, this heuristic is realized by the hashing mechanism, which orders candidate patterns in

terms of the number of buckets which yield them. In this way the sentence "John asked Bill to leave" is generally produced without considering the alternative "John informed Bill that he wanted him to leave". The details of how the fetching scheme accomplishes this will be presented in later sections.

2.3. Language Planning and PHRED

Appelt (1982) has presented language generation as the multi-level process of planning utterances to satisfy multiple goals. A division in this multi-stage process can be made between the task domain and the linguistic domain, or between the system level and the interface level. PHRED operates at the interface level. User input to the UNIX consultant system is first analyzed by PHRAN, producing a conceptual knowledge structure which motivates the system's response (Wilensky, Arens, and Chin, 1984). The planning component of the system exists entirely within the task domain of UC. The UC planner makes the choice of illocutionary act, speech act, and the message to be conveyed. PHRED expresses the message in natural language.

While the ability to handle complex problems in language planning might be desirable even at the PHRED level, it is difficult to perform such planning within a real-time system. It is both counter-intuitive and inefficient to treat language production as primarily a reasoning process involving complex inference mechanisms. In fact, the need for such reasoning in language production seems rare. Thus the UC system draws a convenient, if arbitrary, division between the choices of responses and speech acts made by the UC planner and the lexical and structural choices made by PHRED.

3. The PHRED Knowledge Base

The knowledge base shared by the phrasal analyzer (PHRAN) and phrasal generator (PHRED) consists of pattern-concept pairs, where the pattern contains a linguistic structure and the concept its internal representation. The use of the PC pair as a unit of linguistic knowledge distinguishes PHRED from some other language production mechanisms (McDonald, 1980; Mann and Matthiessen, 1983; McKeown, 1982) in which grammatical information and conceptual information are separated. Associated with each PC pair is a list of properties used for indexing the pair in the database and for adding knowledge to the system, as well as information about forms of agreement among constituents.

This section is intended to provide sufficient detail of PHRED's knowledge representation to make transparent the examples to be covered later. For a more complete discussion of the development and particulars of the pattern-concept pair representation, the reader is again directed to (Wilensky and Arens, 1980). The pattern-concept pair representation as presented here is, with some minor variations, the same as that used in the original PHRAN.

The "pattern" part of the PC pairs is a list of constituents, where each constituent in a pattern is generally described either as a pattern of speech (p-o-s) or as a member of a descriptive category (e.g. person, physical object). Patterns may also be formed by conjunction and disjunction of other patterns and may contain specifications of constraints. For example, the constituent

<and root=remove tense=present voice=active form=infinitive>

is a single-constituent pattern which would generate the infinitive verb "to remove", while

<and p-o-s=noun-phrase <or person physob>>

represents a noun-phrase which refers to a person or physical object.

Patterns may be very specific phrases which refer to particular objects. The pattern

<word=the> <word=big> <word=apple>

represents the phrase "the big apple" used to refer to New York City. This phrase can also be produced by the general pattern

<p-o-s=article> <p-o-s=np2>

when used to refer to an apple.

Idioms are often partially frozen patterns which behave as a particular grammatical unit. The phrase "kick the bucket" behaves as a verb which conjugates but does not passivize. It corresponds to the pattern

<and p-o-s=verb root=kick> <word=the> <word=bucket>

which functions as an intransitive verb.

Part of the knowledge associated with a pattern-concept pair is the correspondence between the properties of the pattern's constituents and the properties of the entire pattern. Associated with the "kick the bucket" pattern above is the knowledge that the person, number and tense of the pattern correspond to the person, number and tense of the first constituent, the form of the verb "kick". In generation, this results in the recursive application of constraints from a pattern to its components: To generate a past-tense verb meaning "died", the system will operate recursively on the pattern above to generate a past-tense form of "kick".

Patterns are not necessarily fixed word-order. For example, in

<person> <root=tell> <person>
<<word=to> <word=get> <word=lost>>

the pattern retains its meaning when passivization is applied or when the subject is deleted. Such patterns are used in combination with *ordering patterns,* which control the various ways in which a pattern may be linguistically realized. An ordering pattern which could be used in conjunction with the "get lost" pattern above is the passive ordering:

<p-o-s=np> <p-o-s=verb voice=passive>
<<word=by> <p-o-s=np>> <#rest>

This pattern is accompanied by the information that the third constituent of the accompanying pattern is the first constituent of the ordering, and the first constituent of the accompanying pattern is the third constituent of the ordering. The extra set of angle brackets here is used to mark the enclosed phrase as being optional to the pattern. The "#rest" indicates where additional constituents are generally inserted.

Thus some patterns have an unspecified word order and do not produce a particular pattern of speech independently. These are combined by PHRED with ordering patterns to allow for idioms or expressions which may appear in various forms, such as "bury the hatchet" in "The hatchet was buried at Appomattox." The same effect could be accomplished without ordering rules by increasing the number of fixed-word-order patterns combinatorially. The use of the ordering patterns, however, has a certain elegance as well as a practical value: it allows the specification of certain idiomatic constructs as relations among particular constituents, regardless of where the constituents appear in the actual output. In this case, the specialized meaning of "telling someone to get lost" is effectively represented by the relationship between the verb "tell" and its complement "to get lost". This meaning may be realized in a variety of forms; for example, the combination of the "get lost" pattern with a passive ordering may produce the sentence "John was told by Bill to get lost."

While there are similarities between the ordering rules used by PHRED and transformational grammar rules, there are some important differences. Most importantly, PHRED assumes no underlying surface structure; rather, the final ordering of a pattern of speech is produced by combining a set of linguistic patterns. Furthermore, there is no strict sequence in which the patterns must be used: A given ordering pattern may be chosen either before or after a pattern with which it is to be combined. In this way PHRED is more flexible than other systems which handle word order as a final phase of the generation process.

The concept part of the pattern-concept pairs used in the PHRAN/PHRED knowledge base have generally been adaptations of conceptual dependency representations (cf. Schank, 1975) with a variety of predicates, quantifiers, and other additions. The representation scheme is intended not to incorporate an irreducible set of primitives or a restrictive syntax, but to provide a functional representation compatible with the UC knowledge base. The programs are flexible with respect to the choice of knowledge representation, and have regularly been used as a tool in testing new representational theories.

In addition to the linguistic patterns and associated conceptual representation, PC pairs contain a set of properties, or attributes, and other information which guides their use. As described earlier, some of this information is used to determine correspondences between a pattern and its constituents. Other properties are used for indexing purposes. There is also a facility for "escapes" or the ability to call a special procedure from within the declarative knowledge representation. While this facility was often exploited in early versions of PHRAN, it is problematic for knowledge bases shared with PHRED. Procedures called during analysis are seldom useful to the generator or *vice versa*. Therefore such procedure calls have seldom been used in PHRED, and an attempt has been made to encode all knowledge in a declarative form which can be used by both the generator and the analyzer.

A simple example of a pattern-concept pair is given below:

```
Pattern: <agent> <root=remove> <object> <<word=from> <container>>

Concept: (state-change (actor ?actor)
                       (state-name location)
                       (from (inside-of (object ?cont)))
                       (to (not (inside-of (object ?cont))))))
```

Properties:
tense=(value 2 tense)
actor=(value 3)
cont=(value 5)
forms=(active-s passive-s)

The above PC pair can be used by PHRED, depending on the concept being expressed, to produce the sentence "You should remove the files from your directory" or the infinitive phrase "to remove a file from the top level directory". The special "value" indicator designates the association of a property of the PC pair with a property of one of its constituents, specified by number. Thus "tense=(value 2 tense)" implies that the tense of the pattern is the tense of the second constituent, the verb. "cont=(value 5)" indicates that the token unified with the variable "?cont" in the conceptual template corresponds to the fifth constituent, the object of "from". The final output is determined by the combination of this PC pair with the input attributes and the ordering patterns selected.

A simple ordering PC pair is given below:

Pattern: <and #3 p-o-s=noun-phrase case=objective>
<and #2 p-o-s=verb form=infinitive voice=passive>
<<word=by>
<and #1 p-o-s=noun-phrase case=objective>>
<#rest>

Properties:
p-o-s=inf-phrase
forms=(passive-s)

The "#2" and "#3" within the ordering pattern indicate that the constraints on the second and third constituents of the coordinated pattern are conjoined with the first and second constituents of the ordering pattern, respectively. The "p-o-s=inf-phrase" property specifies that the pattern produces an infinitive phrase, and the "forms=(active-s)" property restricts the use of this ordering to patterns which have "active-s" among their forms. This ordering pattern could be used, for example, in conjunction with the "remove" pattern above to produce the passive infinitive phrase "the file to be deleted by the user from his top level directory."

4. Implementation

The production of an utterance in PHRED is a recursive process which can be divided into the three phases outlined earlier. *Fetching* is the retrieval of pattern-concept pairs from the knowledge base. *Restriction* consists of validating a potential pattern-concept pair to confirm that it fulfills a given set of constraints and the addition of new constraints to the pattern. *Interpretation* is the generation of lexical items which match the constraints of the restricted pattern.

Each of these phases and its role in the production process will now be discussed in further detail.

4.1. Fetching

While PHRAN and PHRED use the same knowledge structures, the way in which these structures are accessed for the purpose of generation naturally differs from their access by the analyzer. PHRAN must recognize a set of lexemes as possibly corresponding to a pattern and thereby retrieve an appropriate pattern-concept pair from the knowledge base. PHRED, on the other hand, accesses the knowledge base by searching for a conceptual template which fits the concept that is to be expressed.

Because fetching can be a time-consuming part of the generation process, PHRED uses a hashing scheme designed to produce an ordering of candidate patterns with a minimum of computation. As the generator uses the first available appropriate utterance rather than evaluate all potential candidates, this ordering influences the choice process as well as the number of patterns ultimately considered.

Indexing in PHRED is done by multiple-key hashing. Based on the input concept and/or constraints on the desired pattern, the fetching mechanism determines a set of hash "buckets" in which to search for pattern-concept pairs. The buckets are lists of pattern-concept pairs attached to the property lists of specially constructed atoms. These atoms are uniquely determined by a set of attributes; thus the hashing scheme is not susceptible to random collisions due to fullness of the knowledge base. In other words, the computation of a hash key is guaranteed to yield only PC pairs which partially or entirely match the constraints.

Keys may be arbitrary collections of properties and attributes; thus the same indexing scheme used for the most complex patterns is used for simple words. In lexical choices and choices of grammatical constructions, the concept determines the hash keys. For these concepts PHRED constructs hash keys based first on large sets of semantic attributes, then on increasingly smaller sets of attributes. Since a hash into an empty bucket takes very little time, there is no great loss of time efficiency in using a fairly large number of hashes. Although the access to a PC pair through multiple buckets requires some additional space, this space is negligible compared to the size of the knowledge structures themselves.

The construction of hash keys based on successively smaller sets of attributes assures that the PC pairs whose concept most closely matches the input concept will be considered first. The fetching mechanism produces a stream of pattern-concept pairs which are returned one at a time as they are requested by the generator. The rest of the program is insulated from the retrieval process. This way, some of the hashing computation can be postponed until it is required. The use of the stream simulates the predisposition phase of production, as the ordering of the candidate PC pairs in the stream by the hashing mechanism could influence the eventual choice of output.

In the case of the "remove" entry given earlier, the PC pair is indexed according to a combination of the semantic attributes "state-change", "location", "inside-of", and "not-inside-of". This combination is used at the time the PC pair is read in to determine a special atom, such as "i38246953", to which the PC pair is attached. The indexing mechanism ignores variables (e.g. "?actor").

During generation, the fetching mechanism may receive the following input concept:

```
(state-change (actor file1)
              (state-name location)
              (from (inside-of (object directory1)))
```

(to (not (inside-of (object directory1)))))

The atom "i38246953" will be returned as a potential bucket, based on the same semantic attributes as those under which the PC pair is indexed. Other atoms will be produced as well, but will yield empty buckets. Buckets which correspond to more complete sets of attributes are searched first. For example, if the "delete" pattern were constrained to be used only for the deletion of files, it would be retrieved before the "remove" PC pair because the bucket identified by the conjunction of the "file" attribute of file1 with the other semantic attributes of the concept is searched first.

A simple pattern, such as the word "the", does not really have a concept associated with it, and thus is indexed according to sets of its attributes: A search for a definite article would construct a hash key based on the properties "p-o-s=article" and "ref=def" and would thus yield the PC-pair for "the".

A fast, simple method of arranging the order in which patterns are considered is important to the efficiency of the generator. For example, the root of a verb is a much better hash key than its form or tense. But with compound verbs it would be wasteful to retrieve all possible patterns with the correct root without a general, constructive pattern for the appropriate form. Such a search would retrieve "removes" and "removing" before producing "should remove", which is generated from the pattern:

<and p-o-s=verb root=should> <and p-o-s=verb tense=none>

This pattern can be retrieved by hashing on the form and aspect of the verb before hashing on the root alone. To accommodate this type of ordering, the fetching mechanism in PHRED, as in most large databases, may be given information about which keys to use for hashing for a given pattern type. The number of inappropriate patterns which the program considers thus depends on the ordering of the sets of hash keys, which can be specified within the knowledge base. In Spanish, verbs must often be selected to agree with subject, object and indirect object as well, and the speed of the fetching mechanism depends heavily on the number of keys specified.

If no special hashing information is given for a particular PC pair, PHRED will construct hash keys first from the semantic attributes of the concept to be expressed, if any, and the pattern-of-speech (p-o-s).

The fetching component of PHRED, like the other parts of the system, is geared for simplicity and uniformity. In spite of some of the differences among, for example, the selection of a verb, the choice of a referring expression and the selection of an article which agrees with its head noun, the same method is used for fetching in all three cases. The same hashing scheme is employed also to retrieve ordering rules from the database. Such orderings can be effectively retrieved by a hash on their attributes just as any other PC pair can be fetched. For example, a passive ordering can be located both in a hash bucket corresponding to <and p-o-s=sentence voice=passive> and one corresponding to <p-o-s=sentence>. It will therefore be retrieved before an active ordering if the input constraints to the fetching process include the passive voice attribute, but will appear in the same bucket as the active ordering if only <p-o-s=sentence> is given.

The main loop of PHRED passes to the fetching component the set of constraints which a PC pair must satisfy. Typically, if there is a specific phrase, structural formula or other pattern which directly satisfies these constraints, it

will appear in the stream before more general patterns. A pattern of unfixed word order will generally appear in the stream before an ordering rule. In these cases both the pattern and the ordering must be used to produce the desired output. The fetching mechanism is repeatedly called to return patterns from the stream until all possible constraints are satisfied. For example, to produce the phrase "... not to remove the file", a negative ordering, infinitive, and "remove" pattern must all be fetched before the phrase can be restricted. The manner in which these patterns are combined is discussed in the next section.

The fetching component of PHRED constitutes about 10K bytes of object code, one tenth of the total program. A profile of PHRED shows that more than half of the CPU time consumed by the generator is spent in the fetching process. Earlier versions of the program, which did no ordering of candidate patterns in the fetching phases, spent less time fetching but more time overall.

4.2. Restriction

Each time a candidate pattern is returned from the stream by the fetching mechanism, it is passed to the restriction phase, which creates an instance of the pattern, adding new constraints to the pattern constituents while simultaneously verifying that the PC pair meets the constraints given. There are three main aspects of this process: *unification* of the variables within the PC pair's conceptual template and the associated properties with the target properties and concept, *elaboration* of the pattern constituents to include properties from corresponding properties in the pattern indicated by the "value" marker, and *combination* of the properties of constituents among the pattern and ordering patterns.

The following is an example of an instance of the "remove" PC pair given earlier after restriction:

```
Pattern: <and object concept=file1 p-o-s=noun-phrase case=objective>
         <and root=remove form=infinitive voice=passive>
         <<word=by>
          <and agent concept=user1 case=objective>>
         <<word=from>
          <and container concept=directory1 case=objective>>

Concept: (state-change (actor file1)
                       (state-name location)
                       (from (inside-of (object directory1)))
                       (to (not (inside-of (object directory1))))))

Properties:
    p-o-s=inf-phrase
    tense=(value 2 tense)
    actor=file1
    cont=directory1
    forms=(passive-s)
```

This PC pair is the product of applying the restriction process twice in succession, once to the passive infinitive ordering and once to the "remove" pattern.

Unification has occurred to bind the variables "?cont" and "?actor". Elaboration has added the tokens bound to these variables to the individual constituents. Combination of the various patterns has produced a pattern whose constituents are specified by the conglomeration of constraints of the PC pairs used.

Any of these three aspects of the restriction phase may result in failure. In the above example, unification would fail in an attempt to bind the multiple occurrences of "?cont" to different tokens. Elaboration results in failure if a property to be added to a constituent does not fit the other properties. For example, if "directory1" in the example is not a container, the pattern would be judged inappropriate. Combination could likewise result in failure if the constraints from the ordering rule were incompatible with those from the "remove" pattern, for example if it had no passive form.

Properties marked by "value" in the PC pair are treated as variables and unified along with the other properties. If these variables remain unbound throughout the restriction process, however, the pattern retains the property with its "value" marker. This is necessary for future stages of the production process to obtain the property on demand. For example, a noun-phrase pattern in Spanish, where there is gender agreement between the subject of a passive infinitive phrase and the past participle, maintains the "gender=(value 2)" property to reflect that the gender of the NP is the gender of its NP2. This property is not determined until the head noun is chosen, after which it can be retrieved through the NP if necessary.

Restriction uses about 60% of the code of the generator and most of the CPU time not consumed by fetching. The bulk of this time is spent doing repeated unification when a large number of patterns are required.

4.3. Interpretation

The third major phase in PHRED is interpretation, the application of constraints to a restricted pattern to produce a surface structure suitable for output.

The process of interpreting a given constituent may have three possible results: (1) the successful completion of an element of surface structure, (2) the recursive application of the fetch-restrict-interpret sequence on the given constituent, or (3) failure because of the inability of the generator to produce a specified pattern of speech.

The first result occurs when the pattern provides a complete specification of a word or words for output, such as "the big apple", which is specified by the pattern

<word=the> <word=big> <word=apple>

The second case occurs if a constituent contains a more general set of constraints, for example,

<and p-o-s=verb root=remove tense=past>

which requires another recurrence of the fetch-restrict-interpret sequence.

In the third result, where no output produces the desired pattern of speech subject to the constraints given by the uninterpreted pattern, the system must back up to select an alternate pattern. To be efficient, the system must utilize as much as possible the patterns which have already been selected. If the

constituent which fails in the interpretation phase is optional to the pattern to which it belongs, it is deleted. Otherwise, failure results in backing up to the level where the failed pattern was fetched, getting another pattern from the stream, and attempting restriction of the new pattern. Most often this new pattern will be an ordering rule, and most of the failed pattern will be used in the restriction of the ordering pattern. A simple case of this is where the generator fails to produce a pattern of speech for the subject of a sentence and instead generates a passive sentence. In this case the restricted version of the PC pair as it was before the combination with the active ordering pattern is backed up on a stack so that the passive ordering can be tried.

Failure during interpretation is rare, and generally results from an insufficiency of the knowledge base in producing a reference. Although the back-up algorithm employed in such failure is time-consuming, it increases the likelihood that some successful utterance will be produced.

The agreement of constituents within a pattern is assured during the interpretation phase. A constituent which must agree with another has a form such as the following:

```
<and p-o-s=verb root=remove tense=present
  number=(matches 1) person=(matches 1)>
```

This specifies a past tense form of "remove" that matches its subject in person and number. Interpretation results in the substitution of properties from the matched constituent to produce, for example,

```
<and p-o-s=verb root=remove tense=present
     number=singular person=third>
```

In English agreement occurs relatively infrequently. There are few examples where it passes from right to left, such as in subject-aux inversion where the verb agrees with a subject which follows it. In other languages agreement within a pattern may be much more complex. In the Spanish example, "Juan les habló a sus amigos" ("John spoke to his friends") the indirect pronoun "les", which precedes the verb, agrees with the indirect object, which follows the verb.

In all cases PHRED can ensure proper agreement if some order of interpreting the constituents allows the correct application of constraints. The surface order of the constituents is the default order for their interpretation, but interpretation of a constituent where necessary is done only after that of constituents with which it must agree. In English nouns within noun phrases are interpreted before their attached determiners; in more inflected languages verbs must generally be produced last.

Anaphora are handled specially during interpretation. In the case of constituents for which PHRED has already produced references, the generator applies a set of heuristics which will remove the constituent entirely if it is not necessary to the utterance, pronominalize, or regenerate the entire constituent. The principal heuristics are (1) If the anaphoric constituent is optional, remove it from the current pattern, and (2) pronominalize other anaphoric constituents wherever possible. There are of course many cases in which an alternative reference would be preferable, but the method used by PHRED is generally effective in producing coherent references. The heuristics lead, for example, to the production of "Mary was told by John that he wanted the book to be given to him" rather than "Mary was told by John that John wanted the book to be given to John by Mary".

The interpretation mechanism occupies about 20% of the code of the generator, and requires a small amount of time relative to the rest of the program.

5. Other uses of PHRED

Restricted domains such as that of the UNIX Consultant provide only limited exercise for some of the features of the generator, among these the mechanism which produces anaphoric references as described above. Aside from its principal function in the UNIX Consultant system, PHRED has been tested with simple sentence paraphrasing and translation tasks, in conjunction with PHRAN. Below are some examples of the responses produced:

Input: John had to tell Mary that a friend kicked the bucket.

Paraphrase: John had to tell Mary that a friend died.
Translation: Juan tuvo que decirle a Marîa que un amigo se murió.

Input: John had to tell Mary that the bucket was kicked by a friend.

Paraphrase: John had to tell Mary that a friend kicked a bucket.
Translation: Juan tuvo que decirle a Marîa que un amigo le pateó a una cubeta.

Input: John told Mary to get lost.

Paraphrase: John asked Mary to leave.
Translation: Juan le dice a Marîa que él quiso que ella se fuera.

This output is produced simply by having PHRAN pass the conceptual form it produces to PHRED. A much more sophisticated mechanism would be needed to produce coherent paraphrases and translations involving anaphora.

6. Comparison with Other Systems

PHRED differs in design from most other natural language generation systems because of its conception as a generator to accompany PHRAN as part of a language interface. The use of a declarative knowledge base shared between analyzer and generator has helped to make the system practical and easily extensible. Its simplicity and processing speed have made it well-suited for use in a real-time natural language interface such as in the UNIX Consultant.

Primarily for historical reasons, most research in computational linguistics has focused on rules governing syntax. In language analysis, it is often practical to design systems whose principal function is to apply and test such rules by determining the grammaticality of the input. Such systems generally use compositional rules, if any, for determining the semantic content of the input. The task of language generation, however, is inextricably tied to the *appropriateness* of the linguistic output as well as to its grammaticality. Because of this, work in

generation focuses not on the representation of core syntactic rules but on the means by which a *choice* is made among syntactic and lexical constructs. Compositional rules generally fail to constrain this choice adequately. For this reason systems which are designed for language generation have often employed either special choice systems of the type found in systemic grammar (Halliday, 1968), or have had pattern-based grammars of the type found in PHRAN/PHRED and in unification grammar (Kay, 1983), which require a sophisticated mechanism for dealing with the interaction of the patterns. Thus PHRAN/PHRED is the first interface in a natural language-based artificial intelligence system to use a common representation and knowledge base for linguistic knowledge employed in both analysis and production.

The pattern-concept pair representation differs on the surface from traditional grammars because the grammar is embedded implicitly in the knowledge structures. These knowledge structures often require the combination of a number of patterns to produce an utterance. In this way the representation is comparable to unification grammar, which contains patterns associated with functional descriptions. The restriction process described in this paper is similar to the unification procedure in TELEGRAM (Appelt, 1983), which employs a unification grammar.

One difference between PHRED's knowledge structures and those in unification grammar is that conceptual attributes of the PC pairs, as well as functional attributes, or properties, are used to constrain a pattern. Unification grammar, like most feature systems, generally fosters the separation of conceptual and functional components. Another distinction is that, in unification grammar, the syntactic category is given special status; in pattern-concept pairs it is treated as an attribute, and does not necessarily have to be specified for every pattern. This is important for patterns which can be used in conjunction with many different orderings to produce a variety of syntactic structures.

A general difference between the PC pair and other representations lies in the level of specificity of the patterns. The PC pair makes it easy to encode specialized phrases and constructs to be used by the generator. It allows the generator to apply the same mechanisms to both general and specific constructions, and to choose PC pairs based on their conceptual attributes.

The robustness of natural language output is especially enhanced by the pattern-concept pair representation. Much of the knowledge used to produce language, particularly in specialized domains, is specialized knowledge. Some natural language programs treat grammatical constructions and canned or idiomatic phrases independently of "core" grammar and require special rules and procedures to make use of such phrases. In PHRED specialized constructs are selected and produced using the same mechanism as the more productive constructs, facilitating the interaction of linguistic knowledge of varying levels of generality. In this way a wider range of appropriate utterances may be produced from a given conceptual form.

Semantic grammar (Burton, 1976) is another representation scheme which, like that of PHRED, facilitates the use of semantic attributes in language processing. There are versions of such grammars which allow for varying degrees of interaction between syntax, semantics, and pragmatics. PHRED differs from true semantic grammars primarily in that it facilitates the interaction of the more general patterns with the more specialized. Semantic grammars are often too constrained to be adapted to a new domain. Many the knowledge structures in PHRED, by comparison, are general enough so that much of the linguistic knowledge used within the UNIX domain existed in the PHRAN/PHRED knowledge base before UC was even conceived.

Many language generation systems used in conjunction with large programs separate the linguistic knowledge base and lexicon from the conceptual knowledge base of the system (McDonald, 1980; Mann and Matthiessen, 1983; McKeown, 1982). This has the advantage of modularity, allowing for the development and modification of one module without affecting another. It also has the disadvantage of inhibiting the use of conceptual information by the generator, or of requiring redundant representation of such information. In PHRED linguistic knowledge is maintained separately from world knowledge, to permit such advantages as the interchangeability of English and Spanish knowledge bases in UC. However, the generator may access the conceptual knowledge base of the system and such knowledge may interact with the syntactic knowledge. For example, the verbs "remove" and "delete" are synonymous when used to refer to actions on files, but "delete" may not generally be used with physical objects. PHRED restricts the use of "delete" during elaboration by examining the semantic nature of its object. If the object is not a file, the use of "delete" to refer to the action of removing it is prohibited.

A notable difference in implementation between PHRED and other generators is in the fetching mechanism. The division of the choice problem into predisposition and selection components allows PHRED to bias its choice of utterances using a specialized hashing scheme. This has proven a boon for both simplicity and efficiency, as some of the rules which govern choice are carried out by a simple hashing process and thus fewer patterns reach the restriction phase.

PHRED's restriction process is similar to the unification carried out in many other systems. It is complicated, however, by the interaction of general ordering rules with specialized patterns. This requires the repeated combination and unification of constraints obtained from the various patterns.

Other systems, such as Penman (Mann, 1983), and TEXT (McKeown, 1982) attack the problem of generating coherent multisentential text. This involves the influence of linguistic rules governing reference and focus on the process of deciding what to say. PHRED is not well equipped for this problem. While PHRED produces multisentential text when UC passes it successive concepts to express, it has no knowledge of coherence. Nor is there substantial communication between the PHRED level of production and the higher levels of language planning. Such communication, as described by Appelt (1982), would allow for the generator subsume multiple UC goals. In PHRED and UC much of the process of producing utterances is not considered as planning *per se* but as the application of prestored knowledge about how language is used. The distinction between this prestored knowledge and general planning is analogous to the difference between compiled and interpreted code in programs. More research is required on how knowledge is compiled and on how the use of prestored knowledge about patterns of speech can be used in conjunction with general knowledge about planning.

7. Future Directions

While PHRED is a successful implementation of a fairly efficient, robust, and cognitively realistic generator, it has served also to open up new ground for further work. This work involves aspects of language processing not directly involved in PHRED as well as problems with the PHRED approach and implementation.

7.1. Hierarchical Representation

One deficiency with PHRED as described here is actually a problem with most linguistic knowledge representations. This is the lack of an effective means by which similarities among many linguistic structures can be encoded. The specification of the pattern-of-speech property, such as "p-o-s=verb-phrase" or "p-o-s=noun", provides a great deal of information about the use of a pattern, as does the specification, for example, of attributes such as the voice and form of a verb. However, often the similarity in behavior of linguistic elements cannot be easily determined by the possession of a particular feature. When an arbitrary set of features is used to constrain such behavior, this set of features is redundantly represented among similar linguistic structures. This leads to inefficiency in the knowledge base and the requirement that a great deal of redundant knowledge be encoded for each structure added.

A simple example of the potential inefficiency of the use of features to determine complex linguistic behavior can be found in compound verbs. Such compound verbs as "was eating", "has gone", "is loved", and "kept staring" have a number of similarities in their linguistic behavior, such as the conjugation of the "helping" part of the compound verb, the appearance of adverbials between parts of the compound verb, and the reference of the participle to the event being described. The verbs "was", "has", and "is" as used above act as auxiliaries, as they may appear in inverted form and in elliptical constructs. Examples of such linguistic behavior manifested by auxiliaries appear in the sentences, "Was John eating?", "John has gone, hasn't he?", and "John is loved by his friends and Mary is, too." Other helping verbs do not appear in these forms, as evidenced by the ungrammatical sentences, "Kept John staring?" and "Mary got beaten in the race and Bill got, too." Some helping verbs, such as "dare", may or may not have auxiliary status, depending on the speaker: Either "I dare not go" or "I don't dare go" is acceptable. One can further group helping verbs according to either syntactic or semantic behavior. For example, "John was beaten" and "John got beaten" exhibit numerous semantic similarities, although "get" and "be" differ syntactically.

The common approach to representing the syntactic and semantic information described above is to attach features to the compound verb patterns and to the helping verbs, features which distinguish them from other verbs, although they all share the same syntactic category. Thus a modal auxiliary verb would have the syntactic category of verb, the modal and auxiliary attributes, and attributes indicating its person, number, and tense, among others. Since all modal auxiliaries share a common subset of these features, the representation is redundant. A knowledge base with multiple entries for the verb "dare" could have an entry for the modal auxiliary, a modal entry without the auxiliary feature, and an entry for "dare" as used with the infinitive form of the main verb. In PHRAN/PHRED, as in many representations, each entry repeats a common set of syntactic and semantic features.

Such redundancy can appear in lexical entries or in patterns in PHRAN/PHRED. The pattern-concept pair representation permits the representation of specialized knowledge about phrases and grammatical constructions, but does not really identify the fact that these phrases and constructions are often subclasses of other grammatical units. For example, the term "book marker" in PHRED is treated as a noun, since it behaves syntactically as a noun, but has associated with it specialized knowledge about its meaning. In fact, there are quite a number of noun-noun pairs which are similar ("salt shaker", "door stopper", etc.). While there is certainly some specialized knowledge about what

the term "book marker" refers to, it is redundant to treat each noun-noun pair independently, since some knowledge--at least that the first noun of the pair represents the object of an action referred to by the second noun--is common to many such noun-noun pairs.

An apparent aid in eliminating these redundancies is a hierarchical encoding of linguistic knowledge. In the case of the compound verbs described above, a category for modal auxiliary helping verbs allows a set of syntactic features to be parsimoniously specified by category membership. Multiple inheritance is important, as a verb such as "dare" inherits its lexical and semantic properties from one category and some of its syntactic properties from others. It is often desirable for entries of different syntactic categories, such as nominalizations and their corresponding verbs, to inherit the same semantic attributes. The passive constructions in "John got beaten" and "John was beaten" inherit some of their attributes from the passive verb category and others from a compound verb category. In general, the hierarchical approach leads to a proliferation of categories based on the common syntactic and semantic behavior of linguistic structures.

A hierarchical representation of categories of linguistic structures which exhibit such common behavior has the following advantages:

(1) *It increases representational efficiency.* The behavior of a structure can be encoded more efficiently by representing explicitly the membership in a category than by encoding a complex set of features for each category member.

(2) *It improves extensibility.* The extension of the knowledge base to include new linguistic entities is facilitated by having organizational categories.

(3) *It facilitates the representation of the relationships among syntactic and conceptual categories.* Having a hierarchy of linguistic knowledge allows the pattern-concept pair to be represented as association between an element in the linguistic hierarchy and an element in conceptual hierarchy. This is true of general knowledge, such as "Verbs are used to refer to actions" and specific knowledge, such as " 'kick the bucket' means to die". The hierarchical structure allows the program to identify from the knowledge structures themselves some of the levels of interacting patterns which otherwise would be obtained through repeated calls to the fetching mechanism.

(4) *It is in accord with psycholinguistic evidence.* From the hierarchical organization of linguistic and world knowledge comes the potential of involving categories into the language production task. Human categorization, particularly the notions of basic level and prototypicality (Rosch, 1977), seems to play a substantial role in language use. Ross (1973) argues that syntactic categories have prototype characteristics. Recent research suggests that most grammatical constructions also exhibit prototypes and their properties (Lakoff, 1984).

Work is in progress to develop a hierarchical representation for use in both analysis and generation. An outline of such a representation and its advantages for PHRAN/PHRED is presented in (Jacobs and Rau, 1984).

7.2. Structured Associations

Much of the work on idiomaticity discussed earlier, as well as research on metaphor by Lakoff and others (Lakoff, 1977; Lakoff and Johnson, 1980) has

suggested that there exist a range of underlying *motivations** for many idioms and grammatical constructions, knowledge of which can help govern the use of language. For example, PHRED in its current form has the knowledge that the phrase "kick the bucket" does not passivize but "bury the hatchet" does, without any attempt to represent the motivation for the latter phrase. Knowing that "bury the hatchet" is motivated, i.e., that "bury" refers to terminating and "hatchet" to war, helps to explain the grammatical properties of the phrase. Ross (1981) has suggested that in many cases the variety of forms in which idioms of this type can appear depends on the ability of the noun component of the idiom to function independently as a noun. Passivization, however, seems subject to a more specific constraint; that is, the ability of the noun component of the idiom to *refer*. To take advantage of this knowledge, a representation of the "bury the hatchet" idiom must encode the information not only that the expression refers to making peace, but that the "hatchet" part of the idiom refers to war or to the tools of war.

As another example where motivation might be useful, PHRED now generates "John took a punch from Mary" and "Mary gave John a punch" without representing the common metaphorical derivation of the two sentences. For example, PHRED might have a pattern

<person> <root=give> <person> <striking-action>

to produce the sentence "Ali gave Frazier a punch". This is thus specialized knowledge about "giving" and a potential object. There might also be a pattern

<person> <root=take> <striking-action> <<word=from> <person>>

used to produce "Frazier took a punch from Ali". Similar patterns might exist for "getting a punch" and "receiving a punch". Treating these patterns independently seems cognitively unrealistic, because motivated phrases are in general easier to use and remember, and inefficient, since a more general representation of the "striking as transfer" metaphor might eliminate the need for some of the specialized knowledge about each of the patterns. While knowing the motivation does not obviate entirely the need for specialized knowledge, it can lead to a more parsimonious encoding of the specialized knowledge.

A potential improvement to the PHRAN/PHRED representation, still being explored, is the treatment of knowledge used to associate language and meaning as *structured associations*.† The structured association is an explicit relation between two knowledge structures which also associates their corresponding "components". These components may be *aspectuals* of the two structures or other arbitrarily related structures. A structured association may be used to relate the concept of a striking action to the concept of a transfer, with the patient of the action corresponding to the recipient of the transfer and the actor of the striking action corresponding to the source of the transfer. A structured association might also relate linguistic structures to associated concepts. For example, a structured association might exist between the "book marker" pattern and its conceptual referent, an association which also relates the "book" and "marker" parts of the pattern to their corresponding conceptual components. The

* The term *motivation* is due to Chuck Fillmore and George Lakoff, personal communication.

† The term *structured association* and the use of structured associations in language processing were suggested by Robert Wilensky, personal communication.

"bury the hatchet" expression may likewise be related to a concept by a structured association, with the "hatchet" part corresponding to the "war" part of the concept and "bury" corresponding to the action of terminating the war. Metaphors and pattern-concept pairs alike may thus be represented as types of structured associations.

Often the relationship between two knowledge structures is itself a complex concept, and can be related to other such relationships. Structured associations themselves form a conceptual hierarchy. In this way the specific association between a linguistic structure and a conceptual structure may be a special case of a more general association. The correspondence between agent of an action and source of a transfer, for example, is not unique to the "striking as transfer" metaphor. In fact, many other such metaphors exist, such as in "giving a kiss" and "giving a shot", where the agent of the activity is the source of the corresponding transfer. These metaphorical associations, therefore, are descendents of an "action as transfer" association in the hierarchy. Many of the similarities among the metaphors, such as the correspondence between agent and source and between nominalization and action, need not be explicitly encoded for each metaphor but are inherited from the more general association.

While much of the knowledge about a specific metaphor or pattern-concept association may be inherited from structured associations above it in the hierarchy, specific knowledge is required to *realize* the association. For example, the realization of the sentence "Ali gave Frazier a punch" from its conceptual representation demands special knowledge about which particular associations are applied. The application of a sequence of general associations otherwise could produce "Ali gave Frazier a sock", which fails to convey the same message but rather suggests that Ali gave Frazier one of his stockings. Thus, the generation *process* must make use of general as well as specific structured associations and rely on specific knowledge to "trigger" general associations. In this example, the concept of a punching action triggers the general "action as transfer" association, along with specific associations between the action and the lexical items "give" and "punch".

The structured association bears some similarity to the idea of a "view" (cf. Moore and Newell, 1973; Wilensky, 1984), but is more general. The term "view" is used principally to describe relationships used to understand analogous concepts, while the structured association relates arbitrary knowledge structures. Also, the structured association is not a primitive relation, as structured associations themselves are a conceptual hierarchy.

Gentner's *structure-mapping* theory (Gentner, 1983) addresses problems in understanding analogy which are comparable to some of the metaphorical issues discussed above. Gentner focuses on the process by which structure-mappings are synthesized rather than on the explicit representation of associations which may be used for such mappings.

Incorporating structured associations into a hierarchical knowledge base could further facilitate the interaction of general and specialized linguistic knowledge. Thus PHRED, and PHRAN as well, could gain efficiency in representation from the generalizations which apply without losing the advantages of having specialized patterns.

7.3. Context and Memory Models

Another major area for future work is in the development of models of memory which help account for the role of context in language processing. A

kind of spreading activation model (Arens, 1982) was used in UC to help resolve references and to activate particular goals, plans and speech acts. The idea behind an activation-based model is that subtle changes in context can influence language processing without requiring the addition of large amounts of conceptual information to all of the linguistic knowledge structures.

A spreading-activation model has the potential of being especially useful in the predisposition phase of generation. Information about objects and events which have been explicitly referred to or activated in the current context, as well as about the topic of conversation and the participants in the conversation, can influence the language considered. There are, however, three major practical difficulties with using spreading activation as a means of controlling the effect of context on language production. First, the spreading activation model is a parallel one which tends to produce slow, awkward simulations. Second, the encoding of knowledge into a network suitable for such a memory model must involve either a complex method of acquiring the knowledge from data or a contrived set of associative strengths based on introspection. Finally, while spreading activation is often effective in describing subconscious effects such as associative priming, it is difficult to account for the interaction of such effects with conscious or planned behavior. Most likely, a memory model will prove useful as a means of modelling the predisposition process and will simplify, but not replace, language planning and language selection.

8. Conclusion

PHRED is a practical language generator for use in natural language interfaces. The use of a knowledge base shared with the PHRAN analyzer makes it easy to adapt the interface to a variety of domains in which understanding and production of fairly robust language is required.

In addition to its value as a useful language processing tool, PHRED has paved for the way for better models of language generation and linguistic representation. Currently the PHRED approach is being extended to provide a better account of the role of categories and of metaphors in language use. These models are being explored with the goal of designing cognitively realistic, robust, and efficient natural language interfaces.

9. A Detailed Example

Below is a trace of PHRED while generating the sentence, "Typing 'rm filename' causes the file filename to be removed from the current directory." This is a fairly simple example, but demonstrates well the process used by PHRED to produce an output.

The input to the generator is the concept which the UNIX consultant has chosen to express, in response to a question about removing files in UNIX. The concept represents UC's knowledge that using the 'rm' command is an established plan (here "planfor") for deleting a file.

```
***Fetching***

concept =
   (planfor
      (result
         (state-change (actor file1)
                       (state-name location)
                       (from
                          (inside-of
                             (object current-directory)))
                       (to
                          (not
                             (concept
                                (inside-of
                                   (object current-directory)))))))
      (method
         (mtrans (actor *user*)
                 (object
                    (command (name rm)
                             (args (filename))
                             (input *standard*)
                             (output *standard*)
                             (diagnostic *standard*)))
                 (from *user*)
                 (to *UNIX*))))
```

There are a number of patterns which could potentially be used to express the concept that an action is a plan for something. In this case PHRED selects a pattern with the verb "cause". In examples such as this one, where PHRED's retrieval mechanism does not favor a particular construct, the generator selects at random from the alternatives. After the selection is made, the restriction process is applied to the pattern.

Restricting

```
Pattern:  <p-o-s=act-phrase>
          <and p-o-s=verb root=cause>
          <and p-o-s=inf-phrase voice=passive>

Concept:  (planfor (result (*var* result)) (method (*var* method)))

Properties:
          method=(value 1)
          result=(value 3)
          p-o-s=sentence
          form=(declarative active)
          tense=(value 2)
```

The restriction process here results in the addition of the appropriate conceptual components to the constituents of the restricted pattern. The conceptual content of the first and third constituents, which will produce a gerund phrase and passive infinitive phrase, respectively, have been added. This results from the *unification* of the variables "method" and "result" in the list of properties above and the *elaboration* of the constituents specified by the terms "(value 1)" and "(value 3)" attached to these variables. *Combination* with an active sentence pattern adds the subject-verb agreement, and the restricted pattern enters the interpretation phase:

Interpreting

```
Pattern:
          <and p-o-s=act-phrase
             concept=
                   (mtrans (actor *user*)
                      (object
                         (command (name rm)
                                  (args (filename))
                                  (input *standard*)
                                  (output *standard*)
                                  (diagnostic *standard*)))
                      (from *user*)
                      (to *UNIX*))>
          <and p-o-s=verb root=cause
             person=(matches 1) number=(matches 1)>
          <and p-o-s=inf-phrase
             voice=passive
             concept=
              (state-change (actor file1)
                            (state-name location)
                            (from
                              (inside-of
```

```
            (object current-directory)))
      (to
         (not
            (concept
               (inside-of
                  (object
                     current-directory)
                  )))))>
```

```
Properties:
        tense=value 2
        form=(declarative active)
        p-o-s=sentence
        result=value 3
        method=value 1
        concept= ...
        method= ...
        result= ...
```

At this point the generator has successfully applied the input concept to restrict the surface structure chosen, and recursively interprets this structure, starting with the gerund phrase:

```
***Interpreting***

Pattern:   <and p-o-s=act-phrase
                concept=

                      (mtrans (actor *user*)
                         (object
                            (command (name rm)
                                     (args (filename))
                                     (input *standard*)
                                     (output *standard*)
                                     (diagnostic *standard*)))
                         (from *user*)
                         (to *UNIX*))>
```

Interpreting a simple constituent results in a reinvocation of the fetch-restrict-interpret sequence on that constituent:

```
***Fetching***

concept =

        (mtrans (actor *user*)
           (object
```

```
            (command (name rm)
                     (args (filename))
                     (input *standard*)
                     (output *standard*)
                     (diagnostic *standard*)))
          (from *user*)
          (to *UNIX*)))

p-o-s=act-phrase
```

Since there is no pattern which directly generates a gerund phrase (here "p-o-s=act-phrase") with the given concept, the fetch above yields an ordering pattern which can be used for combination with a flexible pattern to produce the final phrase. Thus another fetch is performed before any restriction is done, this time without the "p-o-s" attribute.

```
***Fetching***

concept=

    (mtrans (actor *user*)
       (object
          (command (name rm)
                   (args (filename))
                   (input *standard*)
                   (output *standard*)
                   (diagnostic *standard*)))
       (from *user*)
       (to *UNIX*))
```

PHRED searches for a way of expressing the transmission of the 'rm' command to the operating system. The hashing mechanism gives preference to the terms for technical transmission of commands, because the concepts associated with these terms match the input concept more closely, but a problematic pattern still results:

```
***Restricting***

Pattern: <person> <root do> <command>

Concept:

        (mtrans (actor (*var* actor))
           (object (*var* command))
           (from (*var* actor))
           (to *UNIX*))


Properties:
      command=

        (command (name (*var* name))
```

```
                (args nil)
                (input (*var* input))
                (output (*var* output))
                (diagnostic (*var* diag)))

        name=(value 3 command-name)
```

This pattern fails during unification because it requires that the command not have arguments, something which the fetching mechanism failed to detect. With the gerund ordering pattern still being saved, the fetching mechanism is called again for another candidate.

```
***Fetching***

concept=

        (mtrans (actor *user*)
           (object
              (command (name rm)
                       (args (filename))
                       (input *standard*)
                       (output *standard*)
                       (diagnostic *standard*)))
           (from *user*)
           (to *UNIX*))
```

The pattern returned here by the fetching mechanism is the next one in the stream after the failed "do" pattern. This new pattern, with the verb "type", is then passed through restriction:

```
***Restricting***

Pattern: <person> <root=type> <command-spec>

Concept:

        (mtrans (actor (*var* actor))
           (object (*var* command))
           (from (*var* actor))
           (to *UNIX*))


Properties:
        command=

           (command (name (*var* name))
                    (args (*var* args))
                    (input (*var* input))
                    (output (*var* output))
                    (diagnostic (*var* diag)))
```

```
command=(value 3)
```

Unification of the variables in the above PC-pair with those in the input concept is followed by elaboration of the constituents and combination with the gerund ordering pattern. This yields the following result:

```
---------------------------------------
***Restricting***

Pattern:
        <and root=type form=progressive>
        <and command-spec
           concept=

                (command (name rm)
                     (args (filename))
                     (input *standard*)
                     (output *standard*)
                     (diagnostic *standard*))>

Properties:
     p-o-s=act-phrase
```

The combination of the "type" pattern with the gerund ordering satisfies the necessary constraints, producing a two-constituent pattern which then proceeds to the interpretation phase:

```
---------------------------------------
***Interpreting***

Pattern:
        <and p-o-s=verb root=type form=progressive>
        <and command-spec
             concept=

                (command (name rm)
                     (args (filename))
                     (input *standard*)
                     (output *standard*)
                     (diagnostic *standard*))>
```

PHRED recursively invokes the interpretation procedure on each of the two constituents, starting with the progressive verb:

```
---------------------------------------
***Interpreting***

Pattern:
     <and p-o-s=verb root=type form=progressive>
```

```
---------------------------------------
***Fetching***

p-o-s=verb
form=progressive
root=type
```

This fetch uses a hash on the root and form of the verb given to retrieve the progressive form "typing", whose properties unify trivially with the given constraints:

```
---------------------------------------
***Restricting***

Pattern:
     <word=typing>

Properties:
     form=progressive
     p-o-s=verb
     root=type
```

```
---------------------------------------
***Interpreting***

Pattern:
          <word=typing>

Properties:      root=type
           p-o-s=verb
           form=progressive
```

The word "typing" and its properties are now completely specified, so no further restriction is needed. The next constituent in the gerund phrase, the noun phrase which describes the command 'rm', is thus passed to the interpretation mechanism:

```
***Interpreting***

Pattern:
     <p-o-s=noun-phrase command-spec
          concept=

               (command (name rm)
                    (args (filename))
                    (input *standard*)
                    (output *standard*)
                    (diagnostic *standard*))>
```

As usual, interpretation first results in a fetch:

```
***Fetching***

concept=

     (command (name rm)
          (args (filename))
          (input *standard*)
          (output *standard*)
          (diagnostic *standard*))

p-o-s=noun-phrase
```

The pattern selected for the command is a specific formula for expressing commands to UNIX, the command name following by its arguments, in quotes:

```
***Restricting***

Pattern:
     <word=|`|>
     <command>
     <p-o-s=args>
     <word=|'|>

Properties:
     p-o-s=noun-phrase
     concept=

          (command (name (*var* command))
               (args (*var* args))
               (input (*var* input))
               (output (*var* output))
               (diagnostic (*var* diag)))
```

```
args=(value 3 name)
command=(value 2 command-name))
```

The formula for producing 'rm filename' is straightforward, and results in very little additional work by the generator:

```
---------------------------------
***Interpreting***

Pattern:
        <word=|'|>
        <and command command-name=rm>
        <and p-o-s=args name=filename>
        <word=|'|>

Properties:
        p-o-s=noun-phrase
        concept = ...
        diag = ...
          ...


---------------------------------
***Interpreting***

Pattern:   <word=|'|>


---------------------------------
***Interpreting***

Pattern:
     <command command-name=rm>


---------------------------------
***Fetching***

command-name=rm


---------------------------------
***Restricting***

Pattern: <word=rm>
```

Interpreting

Pattern: <word=rm>

Interpreting

Pattern:

<and p-o-s=args name=(filename)>

Interpreting

Pattern: <word=|'|>

Having completed the clause "Typing 'rm filename' ", the generator now returns to the highest level of the surface structure to finish the sentence. The next constituent in this surface structure is the conjugated form of the verb "cause":

Interpreting

Pattern: <p-o-s=verb root=cause person=(matches 1) number=(matches 1)>

The interpretation mechanism finds the person and number of the first constituent of the surface structure. Since this is a singular gerund phrase, it has the third person and singular properties. These are then used in fetching the appropriate verb form:

Fetching

p-o-s=verb
root=cause
form=basic
person=third
number=singular

As with "typing", hashing results in the retrieval of the correct verb, and restriction is a simple process:

```
---------------------------------
***Restricting***

Pattern: <word=causes>

Properties:
            tense=present
            person=third
            number=singular
            form=basic


---------------------------------
***Interpreting***

Pattern: <word=causes>

Properties: ...
```

The completed specification here results in the interpretation of the third and final surface structure constituent, the infinitive phrase:

```
---------------------------------
***Interpreting***

Pattern:
      <and p-o-s=inf-phrase voice=passive
        concept=

              (state-change (actor file1)
                      (state-name location)
                      (value (*var* val))
                      (from (inside-of
                              (object current-directory)))
                      (to
                         (not
                            (concept
                               (inside-of
                                  (object
                                     current-directory))))))>


---------------------------------
***Fetching***

concept=

(state-change (actor file1)
            (state-name location)
            (value (*var* val))
            (from (inside-of (object current-directory)))
            (to
```

```
                (not
                   (concept
                      (inside-of (object current-directory))))))
p-o-s=inf-phrase

_____________________________________

***Fetching***
```

The first fetch in this case again brings the ordering pattern second brings the "remove" pattern. The restriction process is applied first to the "remove" pattern:

```
_____________________________________

***Restricting***

Pattern: <person> <root remove> <physob> <<word=from> <container>>

Concept:

        (state-change (actor (*var* rem-object))
                (state-name location)
                (value (*var* val))
                (from (inside-of (object (*var* container))))
                (to
                   (not
                      (concept
                         (inside-of (object (*var* container))
                                    )))))

Properties:
          rem-object=(value 3)
          ...
```

After unification and elaboration of this pattern, the pattern is then combined with the ordering pattern for the passive infinitive phrase. This results in the determination of the final ordering of the constituents, and another round of restriction:

Restricting

Pattern:

<and physob concept=file1>
<and p-o-s=verb root=be form=infinitive>
<and p-o-s=verb root=remove form=perfective>
<<word=from>
<and container concept=current-directory>>

Properties:

subject=?inf-phrase-subject
voice=passive
object=?inf-phrase-object
p-o-s=inf-phrase
form=(passive)

Interpreting

Pattern:

<and physob concept=file1>
<and p-o-s=verb root=be form=infinitive>
<and p-o-s=verb root=remove form=perfective>
<<word=from>
<and container concept=current-directory>>

Properties:

p-o-s=inf-phrase
concept= ...
...

As the interpretation starts with the first constituent of the infinitive phrase, PHRED now must produce a referent for the specified file. To do this, it expands the token "file1" to get the necessary information from its attributes.

Interpreting

Pattern:

<and physob concept=

(file
(size nilstruct)
(form nilstruct)
(location current-directory)
(use nilstruct)
(contents nilstruct)
(extent nilstruct)
(name (filename))
(type nilstruct)

```
                    (owner *user*)
                    (protection nilstruct))>

  Properties:
        concept= ...
        p-o-s=noun-phrase

  ---------------------------------------
  ***Fetching***

  p-o-s=noun-phrase
  concept= ...
```

PHRED uses a structural formula to refer to the hypothetical file:

```
  ---------------------------------------
  ***Restricting***

  Pattern: <word=the> <word=file> <name>

  Concept:

        (file
      (size (*var* size))
      (form (*var* form))
      (location (*var* directory))
      (use (*var* use))
      (contents (*var* contents))
      (extent (*var* extent))
      (name (*var* name))
      (type (*var* type))
      (owner (*var* owner))
      (protection (*var* protection)))

  Properties:
            ref=def
            p-o-s=noun-phrase
            person=third
            number=singular
            name=(value 3 name)
```

This pattern is the default reference for files, which is superceded when more information about a given file must be conveyed. The noun phrase now reaches the interpretation phase, resulting in the simple verification that its constituents are complete:

```
---------------------------------------
***Interpreting***

Pattern:   <word=the> <word=file> <and p-o-s=args name=(filename)>

Properties:
          concept = ...
          ...
          p-o-s=noun-phrase
          ref=def

---------------------------------------
***Interpreting***

Pattern: <word=the>

---------------------------------------
***Interpreting***

Pattern: <word=file>

---------------------------------------
***Interpreting***

Pattern: <and p-o-s=args name=(filename)>
```

Having completed the reference, the system now continues with the infinitive phrase. The second constituent of the infinitive phrase is the infinitive of the verb "be":

```
---------------------------------------
***Interpreting***

Pattern: <and p-o-s=verb root=be form=infinitive>
```

As with the other verbs, fetching yields the appropriate form:

```
---------------------------------------
***Fetching***

p-o-s=verb
root=be
form=infinitive
```

Restricting

Pattern: <word=to> <word=be>

Properties:
p-o-s=verb
root=be
form=infinitive
voice=active
tense=present

Interpreting

Pattern: <word=to> <word=be>

Properties:
...

The third constituent of the passive infinitive phrase is the past participle of the verb "remove", which is interpreted next. This process similarly results in the completed verb form:

Interpreting

Pattern: <p-o-s=verb root=remove form=perfective>

Fetching

p-o-s=verb
root=remove
form=perfective

Restricting

Pattern: <word=removed>

Properties: p-o-s=verb
root=remove
form=perfective

Interpreting

Pattern: <removed>

Properties: ...

The final constituent of the infinitive phrase and of the sentence is the optional prepositional phrase specifying from where the file is being deleted. The extra angle brackets in the pattern below indicate to the interpretation mechanism that if it fails to produce a reference or if the reference in the prepositional phrase is anaphoric, the entire constituent may be omitted:

Interpreting

Pattern: <<word=from> <and container concept=current-directory>>

The first constituent of the prepositional phrase, the word "from", is already complete:

Interpreting

Pattern: <word=from>

The second constituent, the referent for the "current-directory", is interpreted next:

Interpreting

Pattern: <and p-o-s=noun-phrase container concept=current-directory>

Fetching

p-o-s=noun-phrase
concept=current-directory
ref=def

Unlike the previous noun phrase, there is no specific structural formula for referring to the current directory. PHRED thus uses a general noun phrase

pattern:

Restricting

Pattern:
<and p-o-s=article consonance=(matches 2) number=(matches 2)>
<and p-o-s=noun number=singular>

Properties:
p-o-s=noun-phrase
person=third
number=singular
...
concept=(value 2)
number=(value 2 number)
person=(value 2 person)
...

Elaboration of the pattern results in a two-constituent pattern to be interpreted, the second constituent of which must refer to the "current-directory" concept.

Interpreting

Pattern:
<and p-o-s=article consonance=(matches 2)
number=(matches 2) ref=def>
<and p-o-s=noun concept=current-directory
number=singular>

Properties: ...

While there is no special noun phrase for referring to the "current-directory" concept, there are special noun constructs. PHRED selects randomly between "current directory" and "working directory" for this reference:

Interpreting

Pattern: <and p-o-s=noun concept=current-directory number=singular>

Fetching

p-o-s=noun
number=singular
person=third
concept=current-directory

The reference selected for the directory is the noun-noun pair "current directory". This is interpreted before the article within the noun phrase, since articles are produced after head nouns to ensure agreement:

Restricting

Pattern: <word=current> <word=directory>

Properties:
concept= ...
consonance=hard
person=third
number=singular
p-o-s=noun

Interpreting

Pattern: <word=current> <word=directory>

Properties: ...

The interpretation mechanism judges the noun-noun pair to be completed, and the final determiner is then interpreted:

Interpreting

Pattern: <and p-o-s=article ref=def number=singular consonance=hard>

```
---------------------------------
***Fetching***

p-o-s=article
ref=def
number=singular
consonance=hard
```

As with verbs, the hashing process of the fetching mechanism yields the appropriate article:

```
---------------------------------
***Restricting***

Pattern: <word=the>

Properties:
        p-o-s=article
        ref=def


---------------------------------
***Interpreting***

Pattern: <word=the>

Properties: ...

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

After the final part of the surface structure is complete, a walk through the surface structure tree is used to produce the final output:

```
Typing 'rm filename' causes the file filename to be removed
from the current directory.
```

10. References

Appelt, D. 1982. Planning Natural Language Utterances to Satisfy Multiple Goals. SRI International: AI Center Technical Note 259.

Appelt, D. 1983. Telegram: A grammar formalism for language planning. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics,* Cambridge, Massachusetts.

Arens, Y. 1982. The context model: language and understanding in context. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society,* Ann Arbor, Michigan.

Becker, J. D. 1975. The phrasal lexicon. In R. Schank and B. L. Webber (eds.), *Theoretical Issues in Natural Language Processing.* Cambridge, Mass.

Burton, R. 1976. Semantic grammar: an engineering technique for constructing natural language understanding systems. Bolt Beranek and Newman Report No. 3453.

Chafe, W. L. 1968. Idiomaticity as an anomaly in the Chomskyan paradigm. *Foundations of Language 6 (1).*

Chafe, W. L. Integration and involvement in speaking, writing, and oral literature. 1984. in D. Tannen (ed), *Oral and written language.* Ablex, Norwood, N.J.

Fillmore, C. J. 1968. The case for case. In E. Bach and R. Harms (eds.) *Universals in Linguistic Theory.* Holt, Rinehart and Winston, New York.

Fillmore, C. J. 1979. Innocence: a second idealization for linguistics. In *Proceedings of the Fifth Berkeley Linguistics Symposium,* Berkeley, California.

Fillmore, C. J., Kay, P., and O' Connor, M. C. 1984. Regularity and Idiomaticity in Grammar: The Case of Let Alone. University of California, Cognitive Science Working Paper.

Gentner, D., 1983. Structure-Mapping: A theoretical framework for Analogy, *Cognitive Science 7,* pp. 155-170.

Goldman, N. 1975. Conceptual Generation. In R. C. Schank, *Conceptual Information Processing.* American Elsevier Publishing Company, Inc., New York.

Halliday, M. A. K. 1968. Notes on transitivity and theme in English. *Journal of Linguistics 4.*

Jacobs, P. 1983. Generation in a natural language interface. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* Karlsruhe, Germany.

Jacobs, P., and Rau, L. 1984. Ace: associating language with meaning. In *Proceedings of the European Conference on Artificial Intelligence,* Pisa, Italy.

Kay, M. 1983. Unification Grammar. Xerox Palo Alto Research Center Technical Report.

Kempen, G., and Hoenkamp, E. 1982. An Incremental Procedural Grammar for Sentence Formulation. University of Nijmegen (the Netherlands) Department of Psychology, Internal Report 82-FU-14.

Lakoff, G. 1977. Linguistic gestalts. In *Proceedings of the Thirteenth Regional Meeting of the Chicago Linguistics Society,*

Lakoff, G., and Johnson, D. 1980. *Metaphors we Live By.* University of Chicago Press, Chicago.

Lakoff, G. 1984. There-constructions: a case study in grammatical construction theory. University of California, Linguistics Working Paper.

Mann, W. 1983. An overview of the Penman text generation system. In *Proceedings of the National Conference on Artificial Intelligence,* Washington, D. C.

Mann, W., and Matthiessen, C. 1983. Nigel: A systemic grammar for text generation, University of Southern California, ISI Technical Report #ISI/RR-83-105.

McDonald, D. D. 1980. Language Production as a Process of Decision-making Under Constraints. Ph. D. dissertation, MIT.

McKeown, K. 1982. Generating natural language text in response to questions about database structure. Ph. D. thesis, University of Pennsylvania.

Moore, J., and Newell, A., 1974. How can MERLIN Understand? In L. Gregg (ed.), *Knowledge and Cognition.* Erlbaum Associates, Inc.

Pawley, A. and Syder, F. H., 1980. Two Puzzles for Linguistic Theory: Nativelike Selection and Nativelike Fluency. Unpublished manuscript.

Riesbeck, C. 1975. Conceptual Analysis. In R. C. Schank, *Conceptual Information Processing.* American Elsevier Publishing Company, Inc., New York.

Rosch, E. 1977. Human categorization. In Warren, N. (ed.) *Studies in Cross-Cultural Psychology (Vol. I).* London, Academic Press.

Ross, John Robert. 1973. Nouniness. In Osamu Fujimura, ed., *Three Dimensions of Linguistic Theory.* Tokyo, TEC Corporation.

Ross, John Robert. 1981. Nominal Decay. Unpublished manuscript.

Schank, R. C. 1975. *Conceptual Information Processing.* American Elsevier Publishing Company, Inc., New York.

Wilensky, R., and Arens, Y. 1980. PHRAN--A Knowledge-based Approach to Natural Language Analysis. University of California at Berkeley, Electronics Research Laboratory Memorandum #UCB/ERL M80/34.

Wilensky, R. 1981. A Knowledge-based Approach to Natural Language Processing: A Progress Report. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* Vancouver, British Columbia.

Wilensky, R. 1984. KODIAK - A Knowledge Representation Language. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society,* Boulder, Colorado.

Wilensky, R., Arens, Y., and Chin, D. 1984. Talking to UNIX in English: An Overview of UC. *Communications of the Association for Computing Machinery,* June.