PeriPhrase: Lingware for Parsing and Structural Transfer

Kenneth R. Beesley David Hefner
A.L.P. Systems
190 West 800 North
Provo, Utah 84604 USA

Abstract

PeriPhrase is a high-level computer language developed by A.L.P. Systems to facilitate parsing and structural transfer. It is designed to speed the development of computer-assisted translation systems and grammar checkers. We describe the syntax and semantics of this tool, its integrated development environment, and some of our experience with it.

I. Introduction

Up to 80% of the time needed to develop a new language pair for computer translation is spent in writing source-language analysis and transfer programs. The PeriPhrase language and development environment were created to allow a computational linguist to write such programs more quickly, using high-level rules that are easily written, read, and debugged.

The syntax of PeriPhrase was heavily influenced by its predecessor "PHRASE," which in turn borrowed from BNF, rule-based programming languages like PROLOG, and expert systems. There are obvious similarities to PARSIFAL, Marcus' Deterministic Parser, and many other projects. It is perhaps true that few of the individual features of the language originated with us. However, we believe the synthesis of these features together with a very powerful debugging environment to be unique and significant, reflecting the practical needs of computational linguists building large commercial systems.

II. PeriPhrase Syntax

A PeriPhrase program consists of a declarations section followed by one or more rule packets. Each packet contains one or more rules. All the category names, variable names, attribute names, and action names used in the program must be declared, and the possible values for each attribute must be enumerated. As applying rules is a time-consuming process, packets of rules can be activated only as they are needed, either when a program starts or during execution.

Simple PeriPhrase rules are composed of a **pattern** on the left side and a **rewrite** on the right side, separated by a **rewrite operator**.

```
pattern => rewrite.
```

PeriPhrase tries to match the pattern on the data being parsed. If the pattern matches, then the data is restructured or recoded according to the rewrite. One way of looking at rules is to see the pattern as a "before" snapshot and the rewrite as an "after" snapshot.

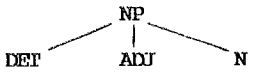
The pattern is composed of one or more **pattern elements**, the simplest being a declared category name. The following are valid patterns:

```
DET ADJ N
V NP
NP VP
```

The most common operation performed by PeriPhrase rules is simple conflation, where all the data items matched by the pattern are made immediate sons under a new father node. The following simple rule forms a noun phrase (NP).

```
! 1   2   3
  DET ADJ N => NP[1, 2, 3].
```

A comment line, preceded by an exclamation mark, is included in this example to highlight the **match units**, which are always counted in strict left-to-right order. In the rewrite, the presence of the category name **NP** indicates the insertion of a node of that category. The square brackets following the **NP** indicate that it is to be a new father node. The numbers appearing in the rewrite are formal pronouns referring back to the match units of the pattern. This rewrite indicates that the first, the second and the third match units (i.e. all the match units) are to be made sons under a new **NP** node, in the order indicated. When the rule fires, a tree like the following will be built.



Many other rules are constructed on the same pattern.

```
V  NP => VP[1, 2].
NP VP => S[1, 2].
```

Because simple conflation is so common, the abbreviation [...], which references all the match units, is provided. The abbreviated rules below are completely equivalent to the rules just described.

```
V  NP => VP[...].
NP VP => S[...].
```

When explicit formal pronouns, rather than [...] are used, the omission of any formal pronoun causes the corresponding match unit to be deleted. The presence of a category name in the rewrite always causes an insertion, either of a new father node or a new terminal node.

Simple conflation rules are much like the context-free phrase-structure rules familiar to formal linguists, but PeriPhrase rules can also be context-sensitive. Suppose that we declared a category **N_V** for marking noun-verb homographs. (It should be emphasized that all category names, and the significance given to them, are determined by the programmer.) When a noun-verb homograph like "walk" occurs in the context "the walk," the following PeriPhrase rule will disambiguate it.

```
! 1   2
  DET N_V => 1 2:=N.
```

That is, if an **N_V** is found immediately preceded by a **DET**, that **N_V** (the second match unit) is recategorized as a noun (an **N**).

Pattern elements can be preceded by a prefix, like the Kleene Star, indicating that zero or more of the indicated items can appear in the data.

```
DET  *ADJ  N  =>  NP[...].
```

Other prefixes available are 1+, which indicates that one or more of the matching items must appear, and 0-1, which indicates optionality.

Similar to the simple pattern elements based on a category name are WILD pattern elements, which will match a data item of any category. The following rule matches whatever is left and forms it into a sentence.

```
*WILD  =>  S[...].
```

It is often convenient to constrain categories by specifying attributes or "features" which must also match or not match.

```
DET(number#plural)  *ADJ  N(number=singular)  =>

                          NP[...](number:=singular).
```

The pattern element **N(number=singular)** will match only if PeriPhrase finds an item of category **N** whose **number** attribute is equal to 'singular.' The pattern element **DET(number#plural)** will match only if the item is of category DET and the **number** attribute of the item is NOT equal to 'plural.' The **:=** or assignment operator in the rewrite indicates that an attribute is to be set to a particular value. The rewrite **NP[...](number:=singular)** indicates that an NP is to be built up in the way already described, and the **number** feature of the overall **NP** is to be set to 'singular.' Attribute restrictions can be set for any simple pattern element in the pattern, and attribute settings can be specified for any inserted or pronoun-referenced item in the rewrite. The = and # signs can be iterated.

```
DET(number=plural=both) ! either 'plural' or 'both'
DET(number#plural#both) ! neither 'plural' nor 'both'
```

When a pattern is being matched, variables can be "loaded" with the attribute values of items being matched. For example, the following pattern would cause variable **X** to be loaded with the value of the **number** attribute for the **N** and the variable **Y** to be loaded with the **case** value.

```
DET  *ADJ  N(X:=number, Y:=case)  =>

          NP[...](number:=X, case:=Y).
```

Inside a rewrite, attributes can also be set from loaded variables, as in the example above, where the number and case of the head noun of a noun phrase are effectively passed up to the noun phrase itself.

PeriPhrase also provides pattern elements more exotic than category names and WILD. An OR pattern element, enclosed in curly brackets, matches when one of an enumerated set of possibilities is found. An exclusion pattern element, enclosed in angle brackets, matches when none of an enumerated set of possibilities is found.

```
{DET | ADJ | N}      ! OR pattern element
<N & ADJ>            ! exclusion pattern element
```

As it is sometimes convenient to specify co-occurence of patterns within patterns, PeriPhrase provides the subpattern, whose elements are bounded by parentheses. The following example assumes that we have declared a category **COMMA**, which would be assigned to the punctuation mark of the same name. The second pattern element will match zero or more instances of the subpattern **(ADJ 0-1COMMA)**.

```
!    1          2            3
   DET  *(ADJ  0-1COMMA)  N  =>  NP[1, 2, 3].
```

Most powerful of all are the hierarchical pattern elements, which allow rules to match whole trees and subtrees that have been built up previously during the analysis. The following example will match an **NP** which consists of a **DET**, an **ADJ**, and an **N**.

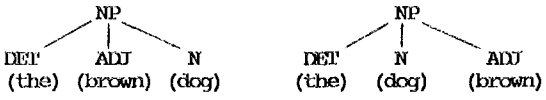
```
! 1  2   3  4
  NP[DET, ADJ, N]
```

III. Transfer with PeriPhrase

The transfer operations of insertion and deletion have already been mentioned. Transfer can also involve reordering and restructuring. The following rule is a simplified example of reordering for transferring from English, where adjectives generally precede the noun they modify, to French, where the adjectives generally follow the noun. Note the reordering of the third and fourth match units.

```
! 1   2     3   4
  NP[DET, *ADJ, N]  =>  1[2, 4, 3].
```

The following trees show sample data before and after this rule has fired.



IV. Actions

It was recognized from the beginning that PeriPhrase itself could not do everything and that it should not try to do everything necessary for analysis and transfer. To accommodate the need to integrate lower-level code, PeriPhrase allows the user to call **actions**, arbitrarily complex C programs, during PeriPhrase processing. Actions appear optionally in rules, both after a pattern and after a rewrite. The following is a sample rule containing action calls to **check_flag**, **print_message** and **set_flag**.

```
DET  *ADJ  N; check_flag(X)  =>
     NP[...]; print_message, set_flag(Y, Z).
```

Actions can also be called as packets are entered and exited. Constants and variables (**X**, **Y** and **Z** in the example above) can optionally be included in action calls as parameters. Because parameters are passed by address, action routines can change the value of variables in the calling PeriPhrase program.

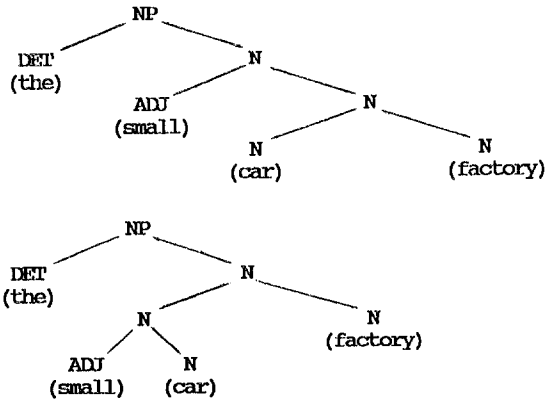
V. Search Order

The rules for each packet are matched left-to-right or right-to-left at the discretion of the programmer. In addition, programmers can optionally specify a traversal order for each packet, either preorder or postorder. If a traversal order is specified, the packet is "free," and PeriPhrase will search down inside tree structures already built up when trying to match rule patterns. Otherwise, a packet is "fixed," and the pattern-matching search is limited to the topmost visible roots of the trees already formed.

VI. Ambiguity and Complex Rules

In most natural languages, especially written English, there are many genuinely ambiguous constructions where an analysis could go two or more ways. For example, the noun phrase **the small car factory** is ambiguous as to whether the writer means a small factory that makes cars or a factory that makes small cars. The analysis chosen will make a big difference if the goal is translation into French or a similar language.

Assuming that **small** is categorized as an **ADJ** and that **car** and **factory** are categorized as **N**s, either of the following trees could be built.



A PeriPhrase programmer might determine that one reading is statistically more common than the other and simply default every time to that one reading. Only one of the following two rules would appear in the grammar, depending on the reading desired.

```
DET ADJ N N  =>  NP[1, N[2, N[3, 4]]].
                         !small (car factory)
DET ADJ N N  =>  NP[1, N[N[2, 3], 4]].
                         !(small car) factory
```

In a similar vein, the analysis could diverge into two parallel paths, and each structure and each analysis would be given a confidence rating. At the end, the analysis with the highest overall confidence rating would win. Another possibility is human interaction.

These three possibilities, statistical defaulting, parallel processing, and interaction, are all responses to the same kind of problem: deciding how to make a genuine choice during analysis. In PeriPhrase, all three possibilities are accommodated by a single specialization, the **complex rule**, which is perhaps the most novel feature of the language. A complex rule lists a set of possible rewrites, one for each alternate path. A rule to handle **the small car factory** structure is the following.

```
DET ADJ N N; action(X)  =>
     choose(X) { NP[1, N[2, N[3, 4]]]   |
                 NP[1, N[N[2, 3], 4]] }.
```

The rewrite section begins with the reserved word **choose**, which takes a "discriminator" variable, here **X**, as an argument. Following **choose(X)** is an OR list of possible rewrites, enclosed in curly brackets and separated by vertical lines. Where n is the value of **X** at the time of execution of the rewrite, the nth rewrite rule in the list is performed.

Usually the variable used to choose a rewrite is set by an action routine in the same rule, but this is not required. Any variable can control the choice, and it could even be a reserved variable set to a desired default reading.

The most straightforward way for an action to set the discriminator variable is to interact with the user. The choices would be presented in some menu form to the screen, and the user's answer would directly choose the rewrite. Actions could also be written to set the discriminator after performing complex syntactic and semantic checks.

Setting the discriminator variable to 0 (zero) causes PeriPhrase to pursue both paths in a pseudo-parallel fashion.

VII. PeriPhrase Development Environment

The PeriPhrase user is provided a development environment which is designed to enhance productivity and shelter the user from irrelevant system-level details. The development environment consists of an editor, an incremental compiler, a source-level debugger, and a user-interface menu.

From the menu, the user can edit any packet, which will be incrementally compiled when execution is restarted. The debugger allows the user to set virtually unlimited numbers of breakpoints on individual rules, packets, and actions. In addition to breakpointing, the user may examine the working memory (database), the production memory (the PeriPhrase source code), PeriPhrase variables, action parameters, and other data relevant to the state of the PeriPhrase program execution. PeriPhrase programs can be "animated." The debugger itself is command-driven and user-customizable, with full macro capabilities.

VIII. Conclusion

We at A.L.P. Systems are finding PeriPhrase to be a valuable software tool for building practical natural-language systems. An earlier version of the language, called PHRASE, is already being used in our translation products as part of the front-end routines that divide a text into sentences. An English analysis program has been started, and we already have a German analysis program with about 600 rules in 80 packets. PeriPhrase is also being used in our Writing Aids division to build a grammar checker for English. We anticipate that PeriPhrase will be used increasingly over the coming years as A.L.P. Systems develops new products and expands its translation line to cover more language pairs. We also expect that PeriPhrase and its development environment will continue to evolve within the established framework.