AFFIX POSITIONS AND COOCCURRENCES:

The PARADIGM Program

Joseph E. Grimes

SUMMER
INSTITUTE OF
LINGUISTICS

UNIVERSITY
OF TEXAS AT
ARLINGTON

[blank]

Affix Positions and Cooccurrences:
The PARADIGM Program

SUMMER INSTITUTE OF LINGUISTICS
PUBLICATIONS IN LINGUISTICS

Publication Number 69

Affix Positions and Cooccurrences:
The PARADIGM Program

Joseph E. Grimes

A Publication of
THE SUMMER INSTITUTE OF LINGUISTICS
and
THE UNIVERSITY OF TEXAS AT ARLINGTON
1983

ISBN 0-88312-095-X

Library of Congress Catalog Card No. 83-50293

Copies of this publication and other publications of the Summer Institute of Linguistics may be obtained from

The Bookstore
Summer Institute of Linguistics
7500 W. Camp Wisdom Road
Dallas, TX 75236

Contents

PART ONE: MORPHOLOGY

PART TWO: COMPUTATION

[blank]

Preface

In presenting this work on morphology I acknowledge a substantial debt to my colleagues Ivan Lowe, Robert A. Dooley, and Gary F. Simons, whose incisive critiques have contributed to clearing away the fuzz that seems to collect around discussions of the internal form of words. I also appreciate the time Robert Croese spent in getting me into the complexities of Mapudungu, and his awareness of the problems posed by hidden sets.

Albert Bickford and Alan and Karen Buseman have played parts in developing the PARADIGM program that I am grateful to acknowledge. Alan L. Teubner has helped me clarify the program documentation. The version described is the one whose designation when it appears on the computer screen is Version 2.209, which was released for testing in August of 1982.

[blank]

Part One

MORPHOLOGY

[blank]

1 Affix Positions

Language gets quite intricate where affixes and function words are concerned. The intricacies include two kinds of things that are more easily worked on with the aid of a computer than in the traditional way. One of them has to do with the linear order of elements. The other has to do with cooccurrence restrictions.

This book addresses the problem of analyzing language when these intricacies are a factor. The first three chapters have to do with the conceptual basis for understanding them.

The other chapters have to do with PARADIGM, a computer program that embodies this conceptual approach to morphological analysis. The PARADIGM program is designed to assist the linguist in sorting out both linear order and restrictions on cooccurrence. It has a positional analysis capability that shows what the linear order of elements is. It also has capabilities for finding distinct subsets of morphemes and specifying the component subgraphs from which their total cooccurrence potential can be worked out.

Linear order

Positional analysis (Joseph E. Grimes, "Positional analysis," Language 43:437-444, 1967) is a way to determine the linear ordering of affix systems. In complex word structures, certain sets of affixes conventionally come in a particular sequence before or after other affixes. (Eugene A. Nida, Morphology: The Descriptive Analysis of Words, second edition, Ann Arbor: University of Michigan Press, 1949, also discusses this, but his method of working it out is not practical for complicated morphologies. P. H. Matthews, Morphology: An Introduction to the Theory of Word-structure, Cambridge University Press, 1974, gives an overview of general problems of word structure without getting into specifics at the level of this book.)

The way an ordering scheme is defined is as follows: in a language that has many prefixes, certain of them come immediately before the stem whenever they occur; they are the first-order prefixes. If any first-order prefixes are present in a form, second-order prefixes come immediately before them; otherwise they come before the stem. There may be several relative orders of prefixes that can be determined on this basis. Huichol of Mexico has fifteen.

Suffixes work the same way: first-order suffixes always come immediately after the stem, and second-order suffixes after first-order suffixes if there are any, otherwise after the stem, and so on. Huichol has five suffix orders besides its prefixes; Southern Bolivian Quechua has at least twelve; Mapudungu of Chile has twenty-eight.

Positional analysis, as expressed in the PARADIGM program, finds these order properties of affixes. Before explaining how it works, however, I should mention some of the complications.

In both prefixation and suffixation, the order classes are relative, because in a given word there might be no reason for affixes of certain orders to be present. This makes it possible for, say, a fifth-order suffix to come right after the stem, or two positions after the stem, and still be in the fifth relative order. It does not imply that a fifth-order suffix always has four other suffixes separating it from the stem. Nor does it imply that a fifth-order suffix in a word that has only two other suffixes between it and the stem thereby slips into the third order. It implies only that if any lower-ordered suffixes are present, they will always come between it and the stem, and that if any higher-ordered suffixes are present, it will always come between them and the stem. The same relativity holds for prefixes.

Some affix systems have spanned orders. For example, a set of prefixes may behave like second-order prefixes in that they can be separated from the stem only by first-order prefixes if at all, but at the same time the only things that can precede them may be, say, prefixes that can be shown on other grounds to be sixth-order prefixes or higher. The order of these prefixes, then, is neither 2 nor 5, but the entire span 2-5, as with sa- and za- in figure 1. They occur before first-order pa- and ta-, but are preceded only by sixth-order ma- and na, and they cannot cooccur with any of the ones in between.

ORDER	6	5	4	3	2	1	0
PREFIXES	ma-	ba-	da-	ga-	ka-	pa-	Stem
	na-	sa- za-				ta-	

Figure 1. Prefix system of six orders with spanning.

There are three complications that arise in performing positional analysis on affix systems. One comes from alternate orderings, where the same affixes may occur in more than one order.

For example, Mapudungu of Chile has the following forms (called to my attention by Robert Croese):

kɨđaw-KE-KA-n
(work-HABITUAL-BY=STAGES-me)
'I always (-KE) work some (-KA)'

kɨđaw-KA-KE-n
(work-BY=STAGES-HABITUAL-me)
'I work some (-KA) anyhow (-KE)'

Alternate orderings like this cannot be handled well by the procedure that discovers orderings. A way of normalizing the data to prevent things like this from upsetting the analysis is given in chapter 4.

The second complication comes from layering, where there is a complex word structure in which one word form is used as the core of another word form. Huichol of Mexico, for example, has words of up to three layers:

pʉ-tí-yʉ.-'ʉkíi.-tʉ-wa
(assertive-something-self-copy-cause-habitual)
'he teaches himself, he is studying something'
(an inflected verb stem)

té-yʉ́.-'ʉkíi.-tʉ-wá-ME-TE.-MAA-MA
(something-self-copy-cause-habitual-NOMINAL-PLURAL-RELATION-POSSESSIVE)
'learners related to him, his pupils' (an inflected noun; *té-* is the plural counterpart of *ti-*)

ME-KAl-té-NÍ-yʉ.-'ʉkíi.-tʉ-wá-me-te.-maa-ma.-TɄ-KÁI.TɄ-NI
(THEY-MODERATED-something-NARRATIVE-self-copy-cause-habitual-nominal-plural-relation-plural-VERBAL-PAST=DURATIVE-CLOSURE)
'they were his pupils' (a verb)

On the inner layer of this last example there is a fairly complicated verb that means 'teach oneself something' or 'learn'. It is nominalized by *-me*, and that noun is pluralized and possessed, giving a second layer. The plural noun is made back into a verb by *-tʉ*, and that verb is inflected as a finite verb in the past tense to form the outer layer. The prefix *ni-* 'narrative mode' belongs to the outer layer even though it is surrounded by inner layer prefixes.

The point to be made here is that it is perfectly straightforward to analyze each of these layers separately, but risky to try to analyze two or more at the same time. Positional analysis is affected most when two affixes that come in one order when in the same layer may appear to come in the reverse order when they are in different layers. Order is relative to a single layer, not to a multilayered word as a whole.

The third complication comes from cyclic data. In the file TCVERB.DAT, included with PARADIGM as test data for the part of the program that recognizes this condition (see chapter 7), there are three suffixes in the verb system of Tucano of Colombia that form a cycle. When A occurs with B it always precedes it, B precedes C, but C always precedes A when both are present. All three cannot occur in the same word, but any two can. Such a cycle inevitably ties up the positional anlysis.

The actual Tucano suffix strings are

STEM-sĩ'rĩ-ti-TENSE
(STEM-want=to-negative-TENSE)

STEM-ti-cã'-IMPERATIVE
(STEM-negative-emphatic-IMPERATIVE)

STEM-cã'-sĩ'rĩ-mi-TENSE
(STEM-emphatic-want=to-3=masculine-TENSE)

or

STEM-nu'cũ-cã'-sĩrĩ-TENSE
(STEM-always-emphatic-want=to-TENSE)

The positional analysis module of PARADIGM helps you form a full picture of the positional relationships by finding all regularities of ordering including spanning for the data from a single layer. Where alternate orderings or cycles within a layer are possible the data have to be normalized to eliminate one of the orderings. Analysis of the normalized data then provides a framework of relative order classes into which the awkward orderings can be fitted. Ways of coping with these complications are discussed in chapter 4.

Function word systems occasionally show the same kind of ordering characteristics as affixes. When they do, PARADIGM can be used to show their ordering too. It is even possible that phonological sequences could be investigated in the same way.

The best way to follow how positional analysis works is to do it by hand. Here are the steps:

The Precedence Matrix

1. On squared paper list all the affixes you can think of down the side. The order in which you list them is arbitrary; the computer lists them in the order in which it comes across them. List the same affixes across the top of the matrix in the same order. If you come across new affixes later, add them both at the side and at the top in the same order.
2. Go through your data one word at a time. Pick each affix in the word in turn and locate its row from the list at the side of the chart.
3. If there is another affix that immediately follows the one you are looking at, find that affix in the list across the top of the chart. (The procedure works identically for affixes that come after the

one you are looking at, immediately or not, but it is less work to put in only the one that follows immediately.)

4. Put a 1 in the chart where the row for the earlier affix meets the column for the later affix.
5. Move over to the later affix and repeat from Step 2. The result is a precedence matrix in which a 1 means that there is some instance of the affix represented by the row followed immediately by the affix represented by the column. A blank, which is a zero in the computer, means that the affix for the row is not followed by the affix for the column.

Predecessor Analysis

1. Set up a predecessor class number of 1 to begin with.
2. Go down the rows of the precedence matrix looking for any rows that have not been assigned a class number. The first time through this includes all rows.
3. If a row contains no 1s, except perhaps for 1s that in an earlier round were tagged in step 6, assign the current predecessor class number to it. The first time through this class number is 1, set by step 1. Every time you reach step 7, the predecessor class number goes up one higher.
4. If there are no unassigned rows, you are finished.
5. If there are unassigned rows, but none of them has any untagged 1s in it, so that step 3 has not been able to apply, then there is an inconsistency in the data. You may have undistinguished homophones or alternate orders or unrecognized effects of layering or cycles. See section 2 on normalizing the data.
6. If you were able to assign the current precedence class number to one or more rows by step 3, go through the columns that correspond to those rows. Change every 1 in each of those columns to a tagged 1. On paper the easiest way to tag a 1 is to put a dot beside it to indicate that it no longer counts as a 1. The computer tags a 1 by changing it to a 2.
7. Add 1 to the predecessor class number and go back to Step 2.

Successor Analysis

The predecessor analysis went through the precedence matrix by rows. The results were the predecessor classes, which begin with the morphemes that precede nothing else and work backwards. Now the successor analysis goes through the precedence matrix by columns. Its results are the successor classes, which begin with the morphemes that are successor to nothing else and work forwards.

1. Erase the dots on the paper. The computer changes all its 2s back to 1s.
2. For analysis on paper, turn the paper 90 degrees and repeat the precedence analysis. For computation, do the precedence analysis substituting "column" for "row" and "row" for "column."

A set of data from Huichol follows to illustrate positional analysis. They are exhaustive, giving all the possible combinations within one group of Huichol verb prefixes. They are the same input as is used for the two other processes and are in the file HCH.DAT of test data, given also in chapter 7:

```
p&   (& represents a high back unrounded vocoid)
p&-ka2
ka1-p&  (ka1 and ka2 are homophones)
ka1-p&-ka2
m&
m&-ka2
ni
ka1-ni
ka1-ka2-ni
m&-ni
m&-ka2-ni
        (A blank line represents absence of
         all these. It is significant for component
         subgraph analysis.)
ka2
ke
ke-ni
```

The list of morphemes in the order they are found is

p& ka2 ka1 m& ni ke

which also identifies the rows and columns of the precedence matrix. That matrix is

```
p&  010000
ka2 000010
ka1 110010
m&  010010
ni  000000
ke  000010
```

The fifth row, which corresponds to *ni-*, has no untagged 1s in it the first time through, since it is all zeros. Step 3 therefore assigns it to Predecessor Class 1. When all the 1s in the fifth column (which also corresponds to *ni-*) are changed to 2s or dotted 1s, then the second and sixth rows become candidates for Predecessor Class 2, because they no longer have any untagged 1s in them, and so on. The computer output is

```
PREDECESSOR CLASS 001
     ni

PREDECESSOR CLASS 002
     ka2
     ke

PREDECESSOR CLASS 003
     p&
     m&

PREDECESSOR CLASS 004
     ka1

   -------------

SUCCESSOR CLASS 001
     ka1
     m&
     ke

SUCCESSOR CLASS 002
     p&

SUCCESSOR CLASS 003
     ka2

SUCCESSOR CLASS 004
     ni
```

Lay the predecessor classes out on a line starting with Class 1 on the right and the highest numbered class on the left--recall that this is what "predecessor" implies in the way relative orders were defined earlier. Below it lay the successor classes out with Class 1 on the left and the highest numbered class on the right, since it works in the opposite direction:

```
                 PREDECESSORS
ka1        p&        ka2        ni
           m&        ke

                  SUCCESSORS
ka1
m&         p&        ka2        ni
ke
```

From this display it is evident that *m&-* spans the two earliest orders and *ke-* the three earliest. Put the two orderings together and number out from the stem in a way appropriate for prefixes. This gives the picture of figure 2, in which *m&-* spans relative orders 3 and 4 and *ke-* spans relative orders 2 to 4. This spanning reflects the observation that *m&-* cannot cooccur with *ka1-* or *p&-*, and *ke-* cannot cooccur with *ka1-*, *p&-*, or *m&-*.

```
ORDER    |  -4   |  -3   |  -2   |  -1   |  0
---------+-------+-------+-------+-------+-----
PREFIXES | ka1-  |  p&-  | ka2-  |  ni-  | STEM
         +-------+-------+       |       |
         ,      m&-      |       |       |
         +---------------+-------+       |
         |          ke-          |       |
---------+-----------------------+-------+-----
```

Figure 2. Relative orders in a restricted subset of Huichol verb prefixes.

2 Basic Cooccurrences

Two elements in the same order class cannot both occur in the same layer of the same expression. If they were both present, one would have to come before the other, so they could not be in the same order class. (There are exceptions to this in the case of prosodic affixes like tones or nasalization, but these are clearly separable and can either be left out of the main order reckoning or normalized into it, as is explained in chapter 4.)

At the same time, elements in different order classes may be so tightly related to each other in function that the occurrence of one of them in one position blocks the occurrence of another in another position. For example, the object prefixes of Huichol verbs come in four different orders (Joseph E. Grimes, *Huichol syntax*, The Hague: Mouton & Co., 1964, pp. 22-29), but no verb may have more than one object prefix regardless of the order.

ma-ni-ú.-xéiya	'he saw *you*'
ni-*vá*-r-uu.-xéiya	'he saw *them*'
ní-*i*.-xéiya	'he saw *her*' (*i*- eclipses *u*-)
ní-u-*yú*.xéiya	'he saw *himself*'

Elements that cannot cooccur are called *mutually exclusive*. Their relationship can be plotted independently of their positional relations. The two parts of PARADIGM that are concerned with cooccurrence do this.

The traditional idea of a paradigm is related to the same thing: it involves a set of mutually exclusive elements among which a choice has to be made. For example, to use a Spanish verb one has to choose one out of a small number of possible verb endings.

The idea has been broadened by Michael A. K. Halliday in connection with the development of what is now called systemic grammar. (One concise exposition of the systemic idea can be found in the chapter on English grammar in Terry Winograd's *Understanding Natural Language*, New York: Academic Press, 1972.) Systemic grammar recognizes disjoint sets of mutually exclusive elements in language and focuses on two things: the semantic basis of the choices made within a set on the one hand, and the restrictions on those choices that are the consequence of still other choices on the other hand.

Distinct sets

Plotting mutually exclusive elements may be all that is needed to sort out affixes or function words (or a composite of both, as for example Harry Rosbottom found when he sorted out affixes in relation to function words in Guarani of Bolivia: "Different-level Tense Markers in Guarani" *International Journal of American Linguistics* 27:4.345-352, 1961).

Not all mutually exclusive relations are simple, however. The complexity is that a single element may be mutually exclusive with more than one set of other elements for different reasons. That is what gives rise to PARADIGM's special section for recognizing what are called *distinct sets* in among the mutually exclusive relations.

A distinct set is a set of elements that are all mutually exclusive with each other; no two of them can ever cooccur in the same form. If any other element were added to such a set, there would be some pairs of elements in it that would no longer be mutually exclusive.

To call a set "distinct" does not imply that distinct sets are disjoint. In other words, any element in a distinct set may be a member of other distinct sets at the same time.

Thus in the Huichol example just given, the prefix *u-* 'at a reference point' cannot cooccur with *i-* 'third person singular object' because of its position; yet there is nothing about its meaning or syntax that keeps *u-* from cooccurring with any of the other object prefixes. On the other hand, *u-* cannot cooccur with either of the spatial orientation prefixes *na-* 'this side of a reference point' or *nu-* 'that side of a reference point' for reasons that are more clearly semantic, although they are supported by position. The cooccurrence properties of *u-* thus link it with two different sets of prefixes for two different sets of reasons.

That is why it is important not just to list sets of mutually exclusive elements, but to highlight the distinct sets among them and then to show how those distinct sets interrelate. It is not enough for Huichol, for example, to show that there is one set of object prefixes that are mutually exclusive with each other, and another set of orientation prefixes that are mutually exclusive with each other. It is also necessary to recognize the positionally based restriction between *i-* and the orientation prefixes--which shows up in a much more complicated way for *na-* and *nu-* as well.

In PARADIGM the process labelled "distinct sets" does this by showing that the objects and the orientation prefixes form distinct sets that intersect. Each distinct set of morphemes is potentially a linguistically relevant paradigm. However, mere failure of two morphemes to cooccur does not by itself mean that they are part of a significant paradigm, since the failure to cooccur may be due to the appearance or failure to appear of some other morpheme elsewhere. It is for uncovering this kind of restriction that a further analysis in terms of component subgraphs, given later, is needed.

Here is an illustration of how mutual exclusiveness leads to distinct sets of forms. A distinct paradigmatic set of morphemes is *maximally connected*. That is, each morpheme in the set is mutually exclusive with all the other morphemes in the set. Consider four morphemes A, B, C, and D for which the following cooccurrence restrictions hold:

A is mutually exclusive with D
B is mutually exclusive with C and D
C is mutually exclusive with B and D
D is mutually exclusive with B, C, and A

In the example B is mutually exclusive with C and D, and so is C with B and D, and D (ignoring for the moment the relationship it also has with A) with B and C. In other words, the three elements B, C, and D are related to each other in a special way. If we represent mutual exclusiveness--that is, noncooccurrence--between two elements by connecting them with a line segment, then the three elements B, C, and D will be maximally connected with each other--maximally, in the sense that all possible pairs of elements in this subset are actually connected by line segments. In terms of mathematical graph theory, they form a *complete* subset or *clique* within the set of all elements (Lowell W. Beineke and Robin J. Wilson, eds. *Selected Topics in Graph Theory*. London: Academic Press Inc. 1978, p. 6), for which "distinct set" can be considered a linguist's synonym. At the same time, A and D are also mutually exclusive with each other and likewise form a distinct set. There are thus two distinct sets, which in linguistic terms correspond to two potentially paradigmatic sets. They intersect with each other because D is in both:

```
    B
    |\
    | \
    |  C
    | /
    |/
A---D
```

The distinct sets that intersect are (A, D) and (B, C, D):

```
        B
        |\
        | \
        |  C
        | /
        |/
A---D   D
```

The computer groups mutually exclusive forms into distinct sets, or cliques. It also recognizes other morphemes with which a member of a distinct set may be in a mutually exclusive relationship without actually belonging to the same maximally connected distinct set; these other morphemes appear in other distinct sets of their own.

The distinct sets or cliques are the ones most likely to be linguistically important. Their members are usually the elements among which a speaker chooses on semantic grounds, and so they correspond naturally to the idea of a Hallidayan system or paradigm. Intersections of distinct sets such as the one just illustrated, in which D joined with A in one relationship and with B and C in another, are likely to indicate complex cooccurrence restrictions that may require the full apparatus of component subgraphs to analyze.

As in the case of positional analysis, the procedure for finding the distinct sets of mutually exclusive morphemes is the clearest explanation of the process.

The Master Chart

1. On squared paper list all the affixes you can think of down the side. The order in which you list them is arbitrary; the computer lists them in the order in which it comes across them, and the same listing that is used for the positional analysis works here. List the same affixes across the top of the chart in the same order. If you come across new affixes later, add them both at the side and at the top in the same order.
2. Go through your data one word at a time. Pick each affix in the word in turn and locate its row from the list on the side of the chart.
3. For every other affix that cooccurs with the affix in question in the same affix layer, write a 1 where its column meets that row. Do not try to plot affixes between layers--for example, between a verb affix on a stem that gets nominalized and the noun affixes that go with the nominalized form, or other verb affixes if the nominalized form gets reverbalized.
4. Find the same affix at the top of the chart.
5. In that column write a 1 for each other affix in the same affix layer where its row meets that column--the same affixes as in step 3, but in the column as well as in the row.

6. This master chart gives the union of all affix sequences. For each affix it shows what other affixes do and do not cooccur with it. Add to it until you are sure it represents all possible cooccurrences.

The Work Chart

1. Lay another sheet of squared paper over the master chart.
2. Wherever the master chart has a 1, write a 0. Wherever it has a blank, write a 1. The computer does this by simply replacing the master chart with its logical complement.
3. Put 1 into all the squares on the main diagonal to indicate that no affix cooccurs with itself in the same layer. (If an affix gets repeated, treat that differently: treat the process of repetition or reduplication as if it were an affix itself.)
4. Make sure the matrix is symmetric. The square at row I and column J must always equal the one at row J and column I, whatever value I and J may take. Unless your paper is completely opaque, you can double it along the main diagonal and hold it up to the light to check for symmetry.
5. This work chart is the logical complement of the master union chart, with the identity matrix of 1s along the main diagonal added to it from upper left to lower right. It tells what affixes do not cooccur. This is the basis for many important linguistic observations.

Distinct Sets and Intersection Points

1. Go through the work chart one row at a time. The row you are looking at is called the current row. To find intersecting sets you compare the current row with other rows.
2. The 1s in the current row that have not been tagged by step 3 tell you which other rows to compare the current row with. An untagged 1 in the second position of the current row, for example, tells you to compare the whole current row with the whole second row, and so on for all the untagged 1s in the current row except the one on the diagonal. If a 1 in the current row gets tagged by

step 3, do not use it as a pointer to another row. It no longer indicates that a comparison should be made.

3. To compare two rows, go through them position by position. If the current row has an untagged 1 where the other row has a 0, then tag the 1 in the current row by putting a dot beside it (or changing it to 2 by computer) to show that two different distinct sets are involved, and that the element at the head of that column is not part of the distinct set that will be identified from the current row. In other words, its membership in a distinct set appears in some other row, not in the current one.
4. If the other row--the one you are comparing with the current row, not the current row itself--has a 1 or a tagged 1 in it, go on to the next position in both rows.
5. When you have finished, go through the work chart a second time looking at the tagged intersection points that were found in step 3--the dotted 1s or 1s changed to 2s. If row I and column J represents such an intersection point, and if row J and column I, symmetrically, is also an intersection point, double tag both of them by circling both or by changing both 2s to 3s. These represent the intersection points of distinct sets that are masked or hidden because all their members are also included within other distinct sets.

List the Distinct Sets

1. Go through the work chart one row at a time.
2. In each row list the affixes that are represented by untagged 1s. These are the distinct sets of noncooccurring elements within which semantically relevant choices are most likely; they are maximally connected complete subsets or cliques.
3. The work chart may still contain hidden distinct sets, so go through it again looking for any intersection points that were doubly tagged by step 5 of the procedure that recognizes hidden sets. If any rows have doubly tagged 1s in them (circled 1s or 3s, depending on how you have symbolized the symmetric intersection points), copy out a new matrix that consists of only the rows and columns that have a doubly tagged 1 somewhere in them. Eliminate all the rows and columns that contain only 0s, 1s, and singly

tagged 1s (dotted 1s or 2s). Turn all the tagged 1s, dotted or circled, back into plain 1s again. This reduced matrix contains information about distinct sets that was hidden by the pattern of intersections found earlier. Now go back and plot distinct sets and intersection points again in the new matrix and list the distinct sets found at this stage. The process applies recursively until no more doubly tagged intersections are found.

The work chart for the Huichol data already given from the test file HCH.DAT in chapter 7 is as follows. It has the identity matrix of 1's along the diagonal, which state that a morpheme does not cooccur with itself, added in.

```
p&  100111
ka2 010001
ka1 001101
m&  101101
ni  100010
ke  111101
```

In going through the first row, the 1s in it direct you (or the computer) to compare it with the fourth, fifth, and sixth rows. The fourth row, however, has a 0 in it where the first has a 1, in the next to last column. You therefore tag that position in the first row by dotting the 1; the computer program changes it to a 2:

```
By hand:      p&  1 0 0 1 1.1
By computer:  p&  1 0 0 1 2 1
```

After you finish the fourth row, the fact that the fifth position in the first row has already been tagged tells you that you don't need to test the fifth row; <u>ni</u>- is already knocked out as a candidate for membership in the same distinct set or clique as the morpheme <u>p&</u>- that corresponds to the first row. It therefore will be assigned to some other distinct set later.

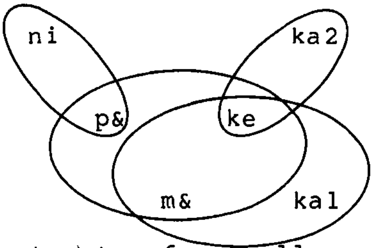
The matrix with its tags is

```
p&  100121
ka2 010001
ka1 001101
m&  102101
ni  100010
ke  122101
```

The results of the analysis into distinct sets of the mutually exclusive morphemes in the example is read off the matrix with tags. For each row, list the morphemes that correspond to the untagged 1s in that row. Where two rows give the same list of morphemes, eliminate the duplicate row:

p& m& ke
ka2 ke
kal m& ke
p& ni

In graphic terms, this is represented by circling the members of each distinct set. This has to be done by hand, based on the list of sets the computer prints out. You may arrange it any way you like; for example:



with four distinct sets of mutually exclusive forms: *p&-* and *ni-*; *p&-*, *m&-*, and *ke-*; *kal-*, *m&-*, and *ke-*; and *ke-* and *ka2-*. Each of the distinct sets intersects with at least one other distinct set; none of them is entirely independent of the others.

The file MAPU.DAT in chapter 7 contains test data that illustrate the hidden sets mentioned in Step 3 of the procedure.

3 Total Cooccurrences

Another process labelled "component subgraphs" provides a way of being completely precise about all the cooccurrence restrictions in a set of data regardless of their cause (Joseph E. Grimes, Ivan Lowe, and Robert A. Dooley, "Closed systems with complex restrictions," Anthropological Linguistics 20:167-183, 1978; Joseph E. Grimes, "A heuristic for paradigms," American Journal of Computational Linguistics microfiche no. 80:51-54 1978). This is a step beyond the analysis into distinct sets. It is needed because the analysis into distinct sets is based only on observations of pairs of morphemes. As a consequence it cannot shed light on restrictions that involve three or more elements at a time. Such multiple restrictions do show up in language, and only an approach like component subgraph analysis can handle them.

The key observation for understanding why this kind of analysis is sometimes needed, as well as for understanding how it works, is to see that when there are two or more sets of choices available in a language, they tend to restrict each other. It is not necessary that all members of one distinct set or clique be able to combine freely with all members of every other to give the so-called Cartesian product of the sets. It is only when paradigmatic sets actually turn out to be completely independent of each other that the information on distinct sets of mutually exclusive forms is all that is needed to analyze systems of affixes and function words.

Component subgraphs

The component subgraph process breaks down information about cooccurrence restrictions into a series of elementary representations called <u>component subgraphs</u>. They show the exact cooccurrence patterns of different portions of the system, and thereby cover every restriction in the whole system. They have to be combined into a master graph, but since that takes more artistic ability and layout ingenuity than PARADIGM is capable of imparting to the computer, the program gives only the elementary component subgraphs that go into the master graph. There are some layout principles for combining the component subgraphs into a master graph in such a way as to make clear all the information that is there as a result of the analysis.

The component subgraphs that PARADIGM produces are the building blocks for stating all the restrictions that occur among distinct sets. PARADIGM looks for morphemes that are related in a straightforward way and puts out the specification for one part of the graph from them. Then it looks at the remainder of the information for another part of the cooccurrence information that it can represent simply and puts that out. This continues until all the information has been put into component subgraph specifications.

The conventions for you to draw component subgraphs from these specifications are simple. Morphemes that are mutually exclusive with each other as far as a single subgraph is concerned appear in the same vertical column between vertical lines or square brackets. Morphemes that can cooccur and hence are not mutually exclusive with each other appear in different vertical columns. That's it.

From here it is a small step to a Hallidayan notation in terms of systems. A paradigmatic set in a vertical column is enclosed between square brackets or vertical lines to indicate mutually exclusive occurrence, which is what in logic is called an "exclusive or" or "disjoint" relation--that is, the choice of one and only one member of the set. Paradigmatic sets in different columns are joined by horizontal lines to indicate conjunction or cooccurrence; that is, elements in one paradigmatic set can cooccur with elements in another. There is no information about order in this kind of analysis, so that you can move columns around into whatever arrangement gives the clearest picture of what is going on.

To get the total cooccurrence picture, you combine the component subgraphs into a master graph by coalescing the parts of them that match each other. There are linked bracket conventions for showing restrictions on cooccurrence when subgraphs are combined. They are explained after the instructions for working through the information by hand are given.

Here are the instructions for finding component subgraphs manually in the same way the computer does it. They are followed by an example from the same Huichol data as were used before.

The Data Vectors

1. On squared paper list all the affixes you can think of across the top. The order in which you list them is arbitrary; the computer lists them in the order in which it comes across them, just as in the preceding two forms of analysis. Do not list them down the side for this analysis. If you come across new affixes later, add them at the top.
2. Go through your data one word at a time. Pick each affix in the word in turn and locate it in in the row across the top of the chart.
3. Make a new row for each word, or for each layer of a complex word. Put a 1 in the row for each affix in the layer. This transforms your data into a form you can work with much more easily than in the raw form.

The Counts

1. Beside each data vector (row), write the number of 1s in that row. This is called the <u>row sum</u>.
2. For each affix (column), add up the row sums that are beside any data vector (row) that has a 1 in that column. Put the total at the bottom of the column. This is called the <u>column sum</u>. Column sums, which are derived from row sums, allow you

to decide what vectors to work on next in a way that makes each component subgraph come out as simple as possible.

The Component Subgraphs

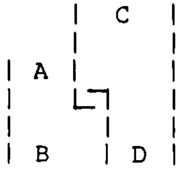
1. Find the smallest column sum that is greater than zero. If several columns have the same sum, pick the first of them. After the first time through, if a column sum has been taken down to zero, ignore that column because it is already out of the picture; the information in it has already been transferred to one of the component subgraphs.
2. Collect all the remaining information on the one morpheme that the column represents by going through all the data vectors, skipping any that have already been checked off. Find each data vector that has a 1 in the column you are looking at, the column with the smallest sum that you chose in step 1.
3. Write down the affix combination that data vector represents.
4. For each column in that data vector that has a 1 in it, subtract the row sum of that data vector from the column sum.
5. Check off the data vector so that you will not look at it again later when you repeat step 2.
6. After you finish the pass through the data vectors, group the affix combinations you wrote down into a component subgraph. Use the following conventions:

   Alternatives, including the absence of certain morphemes, go in the same column and are enclosed between vertical lines.

   Absence of an element is represented by a row of x's.

   Things that can cooccur go in different columns. They may be linked by horizontal lines if that makes it clearer what goes with what.

   Complexities are represented by linking vertical lines, as in the example that follows.



The linked lines in the diagram above show that while A combines with both C and D, B combines only with D, never with C. Looking at it from the other direction, C combines only with A, whereas D combines with both A and B. Following a horizontal path from left to right, the linked vertical to the right of A leads into two options for A, but B does not share both options. Entry to D, however, can come either from the second option with A or from B.

Order means nothing, horizontally or vertically.

7. Go back to step 1 and start through again. Make passes through the remaining data vectors until all have been checked off. If there is a blank row left at the end, make a special subgraph for it.
8. Unite the component subgraphs into a complete graph following the requirements of ordinary logic.

The same Huichol data from the test file HCH.DAT given in chapter 7 that are used to illustrate the sections on positional analysis and distinct sets of mutually exclusive elements translate directly into data vectors. Following each data vector is its row sum, calculated in step 1 of the counts section, and the number of the component subgraph it is assigned to in step 6 of this section. Below the data vectors are the column sums. They are calculated initially in step 2 of the counts section, and revised each time through by subtracting the row sums that are taken out of circulation in step 4 of this section because of being included in one of the subgraphs. For each pass, the low column sum that is eliminated on the next pass is underlined.

The subgraph numbers on the right record which pass takes that data vector out of circulation by incorporating its information into a component subgraph. It is one way of checking off a data vector so that it will not be looked at again; the computer accomplishes the same thing by simply erasing it. The component subgraphs formed from all data vectors that have the same number by step 6 are given below.

OBSERVED FORMS	DATA VECTORS						ROW	SUB-
	p&	ka2	kal	m&	ni	ke	SUM	GRAPH
p&	1	0	0	0	0	0	1	2
p&-ka2	1	1	0	0	0	0	2	2
kal-p&	1	0	1	0	0	0	2	2
kal-p&-ka2	1	1	1	0	0	0	3	2
m&	0	0	0	1	0	0	1	5
m&-ka2	0	1	0	1	0	0	2	4
ni	0	0	0	0	1	0	1	6
kal-ni	0	0	1	0	1	0	2	3
kal-ka2-ni	0	1	1	0	1	0	3	3
m&-ni	0	0	0	1	1	0	2	5
m&-ka2-ni	0	1	0	1	1	0	3	4
xxx (nothing)	0	0	0	0	0	0	0	7
ka2	0	1	0	0	0	0	1	4
ke	0	0	0	0	0	1	1	1
ke-ni	0	0	0	0	1	1	2	1
COLUMN SUMS	8	14	10	8	13	3		1
	8	14	10	8	11	0		2
	0	9	5	8	11	0		3
	0	6	0	8	6	0		4
	0	0	0	3	3	0		5
	0	0	0	0	1	0		6
	0	0	0	0	0	0		7

Component subgraphs are drawn by hand directly from the component subgraph specifications in the PARADIGM program output. xxx stands for the absence of a form.

```
        |ni |         |kal|     |ka2|           |ka2|
1. ke  |   |    2.  |   | p& |   |    3. kal |   | ni
        |xxx|         |xxx|     |xxx|           |xxx|

              |ni |
          |m& |   |            |ni |
4. ka2 |   |xxx|    5. m& |   |    6. ni   7. xxx
          |--xxx--|            |xxx|
```

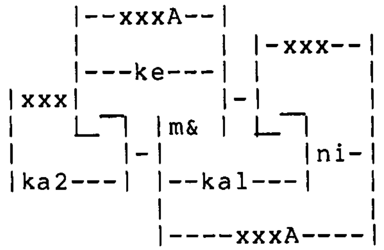
UNIONS OF ELEMENTARY SUBGRAPHS 1 THROUGH 7

At present these have to be made by hand. Matching portions of subgraphs are coalesced to give more comprehensive subgraphs that finally coalesce into a single cooccurrence graph. Tagged blanks like xxxA indicate the same choice topologically speaking, but expressed at two places in the graph. You can think of the graph being twisted around or spindled in three dimensions so that it touches itself at those points that are both labelled xxxA, and then being twisted around again so that it also touches itself at those points that are both labelled xxxB, and so on.

From the set of elementary subgraphs that the PARADIGM program provides, the complete cooccurrence graph can often be built up in any one of several ways. There is a certain amount of artistic leeway about the order in which elements in a column are given, or the order of columns from left to right, or how many times null elements are represented. As you examine your data for meaning as well as form, you should experiment with different layouts for the cooccurrence graph, because some will probably make the meaningful relationships plainer than others. In the process, however, you should constantly watch out that you neither drop affix combinations out of the graph nor add ones that do not actually occur.

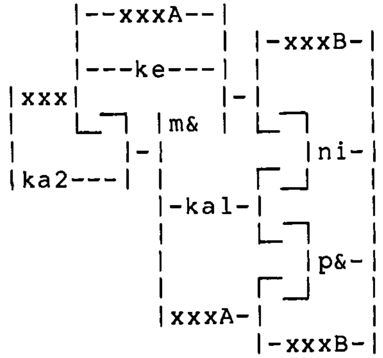
```
   |ni |           |m& | |ni |
8. |   | (6+7)  9. |   |-|   | (5+8)
   |xxx|           |xxx| |xxx|

    |ke |
    |   | |ni |
10. |m& |-|   | (1+9)
    |   | |xxx|
    |xxx|

        |--xxxA--|
        ,        | |--ni--|
        |---ke---|-|      |
    |xxx|        | └──┐   |
11. |   └──┐ |m& |    |   | (4+10)
    |      |-|        |xxx|
    |ka2---| |--xxxA--|   |
```

12.



(3+11)

13.



(2+12)

The complete cooccurrence graph, incorporating all the cooccurrence information in the data vectors and in their corresponding component subgraphs, is in 13. It includes all the distinct sets found by the second procedure in PARADIGM; but it adds to them information about what does not occur (like *ka2-* versus nothing on the left of 13) and information about more than two morphemes at a time (such as the fact that *ka2-* cooccurs with *ni-* only when *m&-* or *kal-* is present also). But it does continue to report the distinct sets *p&-* and *ni-*, *ka2-* and *ke-*, and in a less obvious way, *p&-*, *m&-*, and *ke-*, and intersecting with the last, *kal-*, *m&-*, and *ke-*.

Once the cooccurrence graph is made, the next step is to work out how it matches systems of semantic choices. The means of doing this, together with a discussion of the complications that may arise, is found in my paper "How to Recognize Systems" in William S. Greaves and James D. Benson, eds., *Systemic Perspectives on Discourse*: Selected Theoretical Papers from the 9th International Systemic Workshop, to be published by Ablex Publishing Corporation in the *Advances in Discourse Processes* series.

Part Two

COMPUTATION

[blank]

4 Interpreting the Printout

What comes out of a computer is rarely self-explanatory. Most computational results are an aid to the understanding of something else; they do not by themselves confer understanding or even guarantee that it can be attained. Computers merely handle the data you give them consistently.

Positional Analysis

The positional analysis output begins with the heading you typed in and a page number, and under it the legend

```
POSITION CLASSES

  Predecessor and successor numbering is arbitrary.
  Combine the two to identify spans, then number
the relative orders out from the stem.

      ...  -3  -2  -1  |   0   |   1   2   3   ...
      -----------------+-------+------------------
          Prefixes     | Stem  |    Suffixes
```

Under this comes

```
PREDECESSOR CLASS 001
```

with one or more morphemes underneath it, and the same for each of the other predecessor classes. The numbers for predecessor classes go from the end of the word back towards the beginning.

When all the morphemes have been assigned to predecessor classes, then the process begins again for the successor relationship:

SUCCESSOR CLASS 001

with one or more morphemes under it, then other successor classes for as far as the process goes. The successor classes are numbered from the beginning of the word towards the end.

In case the data are inconsistent, both the predecessor and the successor processes go off the rails at the point of the inconsistency and print an error message:

INCONSISTENT DATA
HOMOGRAPHS? LAYERING? ALTERNATE ORDERS?

Then the computer prints all the morphemes that have not been assigned an order class as Class 000. Although this indicates an incompatible condition in the data, the program does not give up at this point for two reasons. First, seeing how far the computer can organize the positional analysis in both directions helps zero in on which morpheme is causing the trouble. Second, the processes for cooccurrence analysis have nothing to do with the order of elements, so they may be able to do useful work even if the positional analysis has broken down.

Order numbers in the output are probably not the order numbers you want. The reason for this arbitrary numbering is that the program has no way of knowing which order is the stem position.

Once you combine the two orderings as shown in chapter 1 and pick out the position where the stem goes, you should renumber the other positions to give the standard numbering for relative orders, which is out from the stem. In this standard ordering, used at least since Nida's time, Relative Order 1 for prefixes is the first position before the stem, and for suffixes is the first position after the stem. Relative Order 2 for prefixes is the prefixes that come before Order 1 prefixes, and for suffixes is the suffixes that come after Order 1 suffixes, and so on. One way to symbolize this is to call the stem position Order 0, and match its prefixes and suffixes to the negative and positive integers on a number line.

```
... -3 -2 -1  0  1  2  3 ...
```

Within each order the elements line up according to whether they fit only into that order, or whether they begin or end a span at that position.

A schematic display of order relationships including spans can be made from this information. Elements that have the same positional properties are enclosed in the same box. The width of the box shows spanning or single position properties. A complex example of positional relationships in this form follows, adapted from Joseph E. Grimes, Huichol Syntax, The Hague: Mouton & Co., 1964.

```
========================================
-15 |-14 |-13 |-12 |-11 |-10 | -9 | -8 |
----------------------------------------
'aci| nel| kal| p& | ka2| ne2| til| nil|
 'e | pe |---------|    | ma |    |    |
'eci| te |   m&l   |    | tal|    |    |
 'u | zel|   n&    |    |    |    |    |
 ya | wel|  z&ka   |    |    |    |    |
    |    |--------------|    |    |    |
    |    |      kel     |    |    |    |
----------------------------------------
```

```
==========================================
 -7 | -6 | -5 | -4 | -3 | -2 | -1 |  0   |
------------------------------------------
 wa |  r |  e |  i | wal| ne3| ka3| STEM |
 ze2|    |    |  u |    | 'a | kul|      |
    |    |---------|    | ta2| ta3|      |
    |    |    a    |    | yul| ti2|      |
    |    |    +----+    |    | ye |      |
    |    |    |  na     |    |    |      |
    |    |    |  nu     |    |    |      |
------------------------------------------
```

```
========================
  1 |  2 |  3 |  4 |  5
------------------------
 we2| ka4| m&2| kai| me1
zime| ni2| ke2| yu2| ni3
------------------------
       mie      |   t&
       ne4      |  kaku
       n&a      |   ka5
       wa2      |   ku2
                |   me2
                |   yu3
                |   ke2
------------------------
           z&
------------------------
```

A portion of these data from Huichol follow to illustrate the procedure. An important subset of forms is found in prefix orders 8, 11, 12, and 13 in the example just given. The forms that follow, taken from those orders in the total picture, are exhaustive, giving all the possible combinations within this group of Huichol verb prefixes. They are the same input as is used for examples of the other processes, taken from the file of test data included with PARADIGM under the label HCH.DAT.

```
p&   (& represents a high back unrounded vocoid)
p&-ka2
ka1-p&
ka1-p&-ka2
m&
  -ka2
ni
ka1-ni
ka1-ka2-ni
m&-ni
m&-ka2-ni
       (a blank line represents absence of
        all these)
ka2
ke
ke-ni
```

The list of morphemes in the order they are found is

p& ka2 ka1 m& ni ke

which also identifies the rows and columns of the precedence matrix. That matrix is

```
p&  010000
ka2 000010
ka1 110010
m&  010010
ni  000000
ke  000010
```

The fifth row, the one for *ni-*, has no unchanged 1s in it the first time through, since it is all zeros. It is therefore assigned to Predecessor Class 1. When all the 1s in the column that corresponds to *ni-*, the next-to-last one, are changed to 2s or dotted 1s, the second and sixth rows become candidates for Predecessor Class 2, and so on. The program output is

```
PREDECESSOR CLASS 001
     ni

PREDECESSOR CLASS 002
     ka2
     ke

PREDECESSOR CLASS 003
     p&
     m&

PREDECESSOR CLASS 004
     ka1

   ------------

SUCCESSOR CLASS 001
     ka1
     m&
     ke

SUCCESSOR CLASS 002
     p&

SUCCESSOR CLASS 003
     ka2

SUCCESSOR CLASS 004
     ni
```

Putting these two orderings together and numbering out from the stem, which follows the last morpheme since these are all prefixes, gives the complete picture. In it *m&-* spans relative orders 3 and 4 and *ke-* spans relative orders 2 to 4. The order numbering would, of course, be different if all the prefixes had been included in this example.

```
ORDER     |  -4   |  -3   |  -2   |  -1   |  0
----------+-------+-------+-------+-------+-----
PREFIXES  | ka1-  |  p&-  | ka2-  |  ni-  | STEM
          +-------+-------+       |       |
          |      m&-      |       |       |
          + --------------+-------+       |
          |          ke-          |       |
----------+-----------------------+-------+-----
```

Relative orders in a restricted
subset of Huichol verb prefixes.

Distinct Sets of Mutually Exclusive Forms

The information on distinct sets of mutually exclusive forms begins with the heading you typed in and a page number, and under it the legend

DISTINCT SETS OF MUTUALLY EXCLUSIVE FORMS

Under that there are a series of sets that include each morpheme in the data in at least one distinct set. A morpheme may constitute a distinct set by itself. A morpheme may also appear in two or more distinct sets. When it does, those sets are said to intersect. Set intersection indicates that the morpheme or morphemes involved in the intersection are probably involved in two or more different semantic relations.

Each set in the output consists of one or more morpheme names. None of the morphemes in the distinct set can occur with any of the other morphemes listed in the same set.

Each of the morphemes in a distinct set is potentially a member of a single paradigm. Failure of two morphemes to cooccur does not by itself mean that they are part of the same paradigmatic set, since the failure to cooccur may be a restriction that is due to the appearance or failure to appear of some other morpheme elsewhere. It is for uncovering this kind of restriction that the further analysis in terms of component subgraphs may be needed.

The work chart of complemented union vectors for the Huichol data already given is as follows. It has ls added along the main diagonal to show that none of the forms can cooccur with itself.

```
p&  100111
ka2 010001
kal 001101
m&  101101
ni  100010
ke  111101
```

In going through the first row, the ls in it direct the computer to compare the first row with the fourth, fifth, and sixth. The fourth row has a 0 in it where the first has a 1 in the next-to-last position. It therefore changes that position in the first row to a tagged 1; in the computer this is represented by a 2. Then when you go on to examine the fifth row, the fact that the 1 has been tagged tells the computer to skip that test.

The results of the analysis of distinct sets of mutually exclusive morphemes is as follows:

```
p& m& ke
ka2 ke
kal m& ke
p& ni
```

Component Subgraphs

The Huichol data from the file HCH.DAT that are given in the section on positional analysis translate directly into data vectors. Each data vector has a row sum. These do not print out directly, but rather represent internal information that is shown momentarily on the screen as it is being handled. Derived from the row sums are the column sums, which are also internal to the program and so show only momentarily on the screen. They are revised for each subgraph by subtracting the row sums that are taken out of circulation as the preceding subgraph is formed. (Actually the program recalculates them from scratch, which for large quantities of data is not as efficient as subtraction would be, but which allows the computer program to be simpler.)

The computer printout begins with the heading

```
SPECIFICATIONS FOR COMPONENT SUBGRAPHS
```

It then begins to list the subgraphs. Each begins with the name of the morpheme whose column is taken out of circulation by that subgraph, with the heading

```
                    SUBGRAPH FOR x
```

Beneath the heading is the list of forms that go into the subgraph. For the test data in HCH.DAT these are

```
  SUBGRAPH FOR ke
ni ke
ke

  SUBGRAPH FOR p&
p& ka2 kal
p& ka2
p& kal
p&

  SUBGRAPH FOR kal
ka2 kal ni
kal ni

  SUBGRAPH FOR ka2
ka2 m& ni
ka2 m&
ka2

  SUBGRAPH FOR m&
m& ni
m&

  SUBGRAPH FOR ni
ni

----
```

The last line of dashes corresponds to the blank line in the input, which reports that there is a form that lacks any of the prefixes in the set.

The specifications printed in this fashion by the computer are readily converted into cooccurrence subgraphs in the manner explained in chapter 3.

5 Preparing the Data

The data used by PARADIGM are straightforward: strings of morphemes that represent words and phrases. They can be gotten directly from field notebooks or texts. They are taken just as they are found in the data, but you have to separate them by putting hyphens or spaces in between them. It makes no difference in what order the words are presented to the program.

To say the same thing in a way that is technically consistent with the program description, and to clarify a little more what you have to do to prepare them, the data are strings of morphemes that are ordered and normalized. Their ordering within a string is the order in which the morphemes actually occur in the data. In most cases they are strings of prefixes or suffixes, or they may be strings of function words.

Strings may have the morphemes in them represented by their phonological form, like the English suffix -*ing*; or by a prosody, like ` for a low tone that has morphemic value or ~ for nasalization; they may be represented by an inflectional category name like *plural* or by a stem class label like *transitive* or even *stem*.

Since this kind of analysis does not concern itself with specific lexical items, but only with the behavior of affixes relative to each other, it is often best to replace the stem of a word by a general designation, *stem*, or by a word class designation like *verb* or even a type of word class designation like *transitive*. In the Huichol example just given the form *xéiya* 'see' could be replaced by any one of hundreds of other verb stems without affecting the affix relationship, so it could well be replaced by verb or by transitive.

To help trace forms back to the raw data from which they were taken and normalized, PARADIGM incorporates two conventions. The parenthesis convention replaces anything that is written in parentheses in a form by the word STEM. For example, in analyzing Mapudungu

```
(amu)-la-a-I-m-i  becomes STEM-la-a-I-m-i
(mɨtrɨm)-nge-fi-I becomes STEM-nge-fi-I
```

The second convention concerns the use of semicolon. A semicolon and everything after it on a line are ignored. This makes it possible to add glosses or comments in with the data without having them show up as part of the analysis:

```
(feipi)-I ;he said
```

comes through as

```
STEM-I
```

with the gloss stripped away. The test data in the file MAPU.DAT, which are from Mapudungu, are of this form.

Normalization

For PARADIGM to analyze strings of morphemes correctly they have to be normalized in such a way that they meet five requirements for representation.

1. Unique. Homographs--forms that would normally be written the same, either because they are homophones that sound the same or because the spelling tradition of the language fails to distinguish them--must be represented differently. PARADIGM always takes two forms that are spelled alike to be two instances of the same form, and two that are spelled differently--even if the difference is only one of upper and lower case--to be different. Distirct morphemes that sound alike therefore have to be spelled differently somehow.

One way to do this is to represent their meaning; in English, for example, the three enclitics or suffixes that are pronounced -*z* could be represented as -*plural*, -*genitive*, and -*3sg+present* to keep the homophones distinguished.

Morphemes can also be kept distinct by embellishing their phonological representation with extra characters--digits or unused characters or morpheme tags. Thus in the test file HCH.DAT, two Huichol prefixes that have the phonological form *ka-* are distinguished as *ka1-* and *ka2-*. Provided it was done consistently, they could also be represented with some other distinguishing addition, such as *kax-* and *kaz-*, or as *ka=negative-* and *ka=down-*. The important thing is that within one set of data each morpheme be distinct from all the other morphemes even if that distinction is not reflected in the pronunciation.

When homographs are not distinguished, the results of the analysis are invariably wrong. Sometimes the positional analysis section of PARADIGM detects inconsistent ordering when homographs are confounded with each other, because sometimes nonunique representations give the secondary effect of being cyclic (condition 2). This cannot, however, be guaranteed. Whenever homographs are suspected for any reason, go through the set of data with an editing program and decide how to distinguish each instance.

2. *Noncyclic*. The same two morphemes cannot appear in two different orders for PARADIGM's approach to positional analysis. If they do, the program recognizes an inconsistent cyclic condition, prints out everything it has been able to do up to the cycle, prints

```
                  INCONSISTENT DATA
        HOMOGRAPHS? LAYERING? ALTERNATE ORDERS?
```

and goes on to the next section. For cooccurrence analysis cyclic conditions are not a problem because cooccurrence does not depend on ordering.

If there actually are two orders possible between two forms, and they are not a consequence of layering (condition 4), then one of the positions in which one of the morphemes occurs should be treated as if it were occupied by a different but homographous morpheme. For example, if there is a sequence *no-ti* and a separate sequence *ti-...-no* , one of the occurrences of either *ti* or *no* should be treated as though it were a different morpheme; *no-ti* and *ti-...-no2* would be sufficient. After the analysis is finished the two forms can be reunited.

3. *Linear*. When two morphemes are spoken simultaneously, as in the case of a prefix whose tone has separate morphemic value, or a stem with a simulfix such as aspiration or a prosody such as nasalization, these have to be represented one after the other in some consistent way, with a hyphen as separator between them as though they were really in sequence. A prefix *mà-*, for example, where the *ma-* part is an interrogative and its low tone is a completive aspect, could be linearized either as *`-ma* or as *ma-`* (though not both ways in the same set of data).

Infixes also need to be linearized. Many Philippine languages have an infix *-um-* that follows the first consonant of the stem. (Infixes, by the way, are not the same as suffixes that come between the stem and the last suffix. Infixes always interrupt the representation of a single morpheme; they never come between morphemes.) For PARADIGM it is best to linearize such an infix as if it were a prefix, so that a form like *dumagat* is represented as *um-dagat* or *um-STEM*. In the same way, a *y-* prefix metathesized with the first consonant of a stem as in *hyat* is represented as *y-hat* or *y-STEM*.

4. *Unlayered*. Some languages make heavy use of tructures within structures, but PARADIGM is designed to keep track of only one structural layer at a time. (An augmented transition network grammar can keep track of the separate layers, but unfortunately we have to have the kind of information PARADIGM yields before the augmented transition network grammar can be written, so there is a bind involved in using them. See Joseph E. Grimes, ed., *Network Grammars*, SIL Publications in Linguistics and Related Fields 45, Norman, OK: Summer Institute of Linguistics, 1975.)

In the Quechua languages of Argentina, Bolivia, and Peru, and in the closely related Quichua of Ecuador and Inga of Colombia, for example, a verb stem with a few suffixes can be nominalized. The whole thing: stem, suffixes, and nominalizing suffix, then constitutes one structural layer; taken as a whole it behaves as though it were a simple noun. That noun, as another layer, can take the same suffixes as any other noun, simple or complex, which among other things means that it can be verbalized. The resulting verb is a third layer that can take its own suffixes, some of which may be identical with or in positional conflict with suffixes on the verb of the inner layer. There seems to be no theoretical limit on this process of word building by layers, though there are practical limits on comprehensibility if it is carried too far. If the innermost layer of verb suffixes includes some that for a one-layered verb would come only after some of the suffixes in the outer layer, the word-within-a-word structure can make it appear that there is a cycle in the ordering, even when layer by layer there is not.

The strings in a set of data should all come from the same structural layer in order to avoid this confounding. For Quechua it would be prudent to send at least three sets of data through PARADIGM separately: verb suffixes of the outer layer, suffixes to nouns that pay no attention to the internal complexity of the noun stem, but that also do not include the suffixes that belong to the outside layer of verbalized nouns, and suffixes that go with the innermost layer of nominalized verbs. These three analyses need to be brought together at a higher level of abstraction, one that PARADIGM is not intended to handle.

5. *Morphemic*. Allomorphs of a single morpheme need be distinguished only if doing so helps to break an apparent cycle in relative ordering (condition 2). For example, in the Huichol data the prefix *ni-* appears both with high tone and with low tone; high tone is indicated by an acute accent over the first vowel of a syllable, and low tone is indicated by the absence of a tone symbol. The tone of *ni-* is always determined by other factors, so that for this prefix (but not necessarily for other prefixes or for stems) a single representation keeps the morpheme distinct from all other morphemes. Therefore the two allomorphs should be normalized to either the high tone form or the low tone form, but not both.

There is a Quechua suffix that is pronounced -*ra* in some environments and -*ru* in others. For PARADIGM a normalized representation such as -*rU* gives the coherent, one-spelling-per-morpheme representation that is needed both for positional analysis and for cooccurrence analysis, since the distinction between the two phonological shapes of the same morpheme involves neither position class nor coccurrence. It also keeps this morpheme separate from another -*ra* that never changes.

Taking all forms of normalization into account, the Huichol data for analysis given in chapter 2 can be represented as a file containing the following ordered morpheme strings:

ma-ni-u-transitive
ni-va-r-u-transitive
ni-i-transitive
ni-u-yu-transitive

The normalized representation consists of a string of morphemes divided by separators. The separators are hyphen and space. Any other characters that the computer can print, including upper and lower case alphabetic characters, digits, and punctuation marks like the equals sign or quotation marks (but not parentheses or the semicolon because these have special meanings in PARADIGM), can be used to represent the morphemes in the string.

The Huichol morpheme strings just given are normalized in such a way that each morpheme is spelled uniquely (condition 1). They are noncyclic, since there are no apparent alternative orderings (condition 2). They are linear, in that there are no simultaneous phenomena that have separate morphemic status (condition 3). They are unlayered (condition 4), in that only the simple verb structure is covered, even though Huichol does have forms like

kal-táa.-tí-ni.-xéi.yá-mé.-tǘni
'he is the one who keeps an eye on us'

that show three layers very much as Quechua can. (The periods in the representation are for rhythmic breaks inside word structures and can be normalized out.) The strings are morphemic, in that a single representation of each morpheme is all that is used (condition 5). In the case of the stem, a single representation of an entire stem class, "transitive," is all that is used.

Each morpheme string is followed by a carriage return to indicate the end of a line. There are no carriage returns within the line, even after hyphens, since the computer is not limited to the length of a page or even to the width of the screen for the number of characters that it can handle before the carriage return is pressed. (For languages with long strings it is advisable to consult the computer manual about turning off a beep signal that sounds whenever the line gets up over a certain length, and about setting the editor program so that it does not automatically add carriage returns to long lines--set wrap with no number following in the KED editing program.)

6 Running the Program

PARADIGM runs on any computer in the Digital Equipment Corporation PDP-11 series that can handle a PTP program under RT-11. It can be made to run on other computers that can also handle PTP, but they will behave differently in regard to naming files and output devices.

This is computer jargon. It means that to get PARADIGM to work the way this manual says it works, you must first of all have a computer whose operating system--its overall controlling program--is RT-11. In the repertoire of programs that RT-11 can activate it must have the Programmable Text Processor capability, or PTP. PTP is a proprietary product designed by Dr. Gary F. Simons that is available under license from JAARS, Box 248, Waxhaw, North Carolina 28173, telephone (704) 843-2185. It is explained in Simons's _Powerful Ideas in Text Processing_, published by the Summer Institute of Linguistics. The technical description of it is in Simons and Woods, _The PTP Programmer's Reference Manual_ from the same source. This book introduces neither RT-11 nor PTP; it assumes instead that you know how to use both, or that you have someone helping you who understands both. A quick summary of the way to start up RT-11 and PTP is given at the end of this chapter.

The sequence is like this: you have RT-11 start up the PTP system, then have PTP start up PARADIGM. PARADIGM reads your data and does its thing. Once you have done it a couple of times it comes almost without thinking.

The computer needs to be one with a printing device connected to it. Be sure the printer is turned on and has at least ten sheets of paper available.

The instructions for running PARADIGM are divided into three sections. The first covers normal use of the program and assumes that you know how to start up the computer and call up files from it. The second section covers special options. The third is general information about starting and running the computer and naming files that you need if you have not worked with a system like this before.

Normal use

PARADIGM begins by clearing the screen and displaying its own name and version number, needed if anything goes wrong. Then it takes you through a series of questions whose answers give it the information to perform the right set of operations for your needs. Once it finishes its questions and starts to read the data, you can go off and let it work by itself--even computers take time to work when the processes are this complicated--and come back and pick up the results. The only attention you have to give is if you are printing the results and your printing device does not have automatic paper feed. Then you have to feed sheets of paper to it.

(In the examples that follow, anything the computer shows on the screen is underlined and anything you type is not.)

The computer begins by displaying on the screen

```
Output to printer or tape (P or T)? p
```

If you want the results of the analysis printed, which is the normal way to do things, press the letter P, upper or lower case. The computer answers

```
Turn the printer on, then press <RETURN>.
```

Do what it says, making sure the paper in the printer is positioned so that the first line it prints is just below the top edge of the sheet, not at the bottom of the preceding sheet, and the paper feed mechanism is reset to remember that position.

If you want the output to go to tape instead, press the letter T in upper or lower case. The computer will answer

```
Tape for output ready, then press <RETURN>.
```

If you are using tape (in this case, the TU-58 DECtape II digital tape cartridge), you have two or three alternatives, depending on what output devices your computer has and how your tapes are set up. If the tape that contains your input data also has room for the output--you'll have to read up on RT-11 directories to learn how to find out how much space is available, and it should probably hold at least three times as many blocks as are used for the input data--put that tape in the right-hand tape slot. If there is not enough room on the input tape, put a second tape cartridge that you are sure has enough room on it into the left-hand slot (when PTP has the USR set to NOSWAP, if you forgive the jargon, it doesn't require that the system tape be in the left-hand slot until you are finished with the program and want to get back to RT-11). If you have a disk output device or another tape, set that up. Once you have made your choice, press RETURN. The alternatives are

```
Output to? filename.OUT[-1]
           DD0:filename.OUT[-1]
           ??:filename.OUT[-1]
```

The computer asks you where the output is to be directed. The answer you give depends on which of the three options in the last paragraph fits your situation. You have to have a name for the file, regardless of which device it is written on. It is a standard RT-11 file name of one to six alphabetic or numeric characters with no spaces. It could be the same as the input file name, because it is given the file type .OUT to distinguish it from any possible input file.

If the output is to a disk, or to the same tape as the input is coming from, all you need is the file name, the file type .OUT (no space before it), and the [-1] that informs RT-11 that you want the largest available empty space reserved. No spaces. If the output goes to a tape cartridge in the left-hand slot, the file name takes the prefix DD0: with zero and colon, which is RT-11's designation for the left-hand tape slot. If there is some other device available on your computer that you want the results written on, replace the question marks in the third example by the RT-11 code for that device.

If the output from PARADIGM is to go on tape or disk, the next question is skipped because is it relevant only for printout. It is

Type the page heading, then press <RETURN>.
TITLE DATE TIME

Underneath where it says TITLE you type the title you want to see at the top of each printed page. Under DATE and TIME--you get there by spacing over--type the date and time. (The current version of PTP has no way of getting this information from RT-11 automatically, but a later version may.) If you make a mistake while typing, press the <DELETE> key as many times as you need to go back and correct the mistake.

The next questions have to do with the input data. The computer asks

Analyze all morphemes in the data?
Y for yes, N for no:

Type Y, upper or lower case, for "yes" if you want all morphemes included in the analysis, which is the normal way to run PARADIGM. If there are just a few that you want a reading on, however, type N, upper or lower case, for "no." (This yes or no response comes up quite a few times and is handled the same way each time.) The special sequences called up by the "no" answer are given in the next section.

The file you intend to read, the one on which you have placed the data earlier by means of an editing program, goes in the right-hand tape slot if it is on tape:

Disk ready, or tape for input in right-hand slot,
then press RETURN.

If you already chose output to tape and wanted to write the output on the same tape as the input, then the input data are already on the tape in the right-hand slot and all you have to do is press the RETURN key. For any other type of output, make sure the tape containing the input data is in the right-hand slot, and press RETURN.

If the data are not on tape but are in a file on a disk, don't worry about putting anything in the right-hand slot, but press the RETURN key anyway to go to the next step.

<u>Input from?</u> filename.type

This last question designates which of the files on the tape in the right-hand slot, or on the disk, contains the data. The file name consists of one to six characters, and the type is three characters, followed by <RETURN>. The file name and file type are the ones you determined when you created the file using the editor. The only restriction to observe is that the type .OUT is not appropriate for input data, since you could conceivably be trying to use that type with the same file name for output.

Then the computer asks about the parts of the program you want to activate.

<u>Complete processing? Y for yes, N for no:</u>

activates all parts of the program if you answer Y. If you answer N, then it asks about each part in turn as explained in the next section.

After these choices the computer asks

<u>Monitor the processes on the screen?</u>
<u>(WARNING: slow) Y for yes, N for no:</u>

to let you watch it work. To do so may intrigue you, enlighten you, or help you keep awake, but it will also slow down the computation.

<u>Print the internal matrix representations?</u>
<u>Y for yes, N for no:</u>

prints out the matrices of zeros, ones, and tags that the computer works with internally. It isn't much help, but it may help you understand the processes better if you want to take the time.

Unless you have to give attention to feeding sheets of paper into the printer, everything now proceeds under the control of the instructions in the PARADIGM program. The program beeps when it is finished, and the screen returns to the immediate mode of PTP with which it started.

Special use

If you answered N to the question about whether you want all the morphemes in the data analyzed, because you need only a partial analysis, the next question establishes which of the morphemes you want to look at.

```
Is the list of morphemes to analyze on tape?
          Y for yes, N for no:
```

A Y answer for "yes" leads immediately to

```
Input from?
```

to get from you the name of a tape or disk file that contains the list of morphemes. They are separated by spaces and end with <RETURN>, and there is nothing else in the file. For the details of naming a file see the procedure for getting the data source file.

An N answer for "no" assumes you are going to type in the list of morphemes on the spot:

```
Enter the morphemes to analyze, separated by spaces
          and followed by <RETURN>.
```

Type the morphemes as directed. If all you type is <RETURN>, the computer will say

```
No morphemes given in the list to analyze.
          All will be processed.
```

and go ahead and analyze them all.

****

> There is one version of PTP in circulation (Y01) that remembers the directory of the last device it read and looks at its own internal copy rather than at the device itself when given two commands in a row to set up input files. But when you change a tape between inputs, it has no way of knowing what you did, so it thinks it is still on the old tape with the old directory. If you have a list of morphemes to analyze on one tape and your data on another, that version of PTP will say

FILE NOT FOUND. Input from?

because it really hasn't looked at the directory of the tape you just put into the slot. You have to make it behave by first asking it for a nonexistent file on another device, which will stir it out of its lethargy. Answer with

DD0:x (zero, not oh)

That will make it look on the left hand tape for file *x*, which is not there. (WARNING: if you name a tape device as the dummy device to fake out the directory mechanism, make sure there is a tape cartridge mounted in it or the whole program will break down and you will have to start over.) Now quite validly the computer will repeat

FILE NOT FOUND. Input from?

Answer it with the filename.type you really want. Because it has set itself on the directory of the other tape, it will now come back having to read the new tape in earnest, so you will get the result you want. Mind over matter.

****

If you do not choose complete processing of the data, you answer N for "no" to the question about whether you want everything. Then the computer goes through process by process and asks which ones you want to activate:

Are the input data to be printed? Y for yes, N for no:

allows you to bypass having all the strings of morphemes printed out.

Positional analysis (Language 43:437-444, 1967)?
Y for yes, N for no:

asks whether you want the Lister-Grimes method of positional analysis described in the first chapter applied to your data. When you are building up a large set of data you may want to skip this on all but a few critical runs.

Distinct sets? Y for yes, N for no:

asks whether you want the distinct sets of mutually noncooccurring elements printed after the data have been read. This may be sufficient to allow you to pinpoint paradigmatic relationships, or it may not.

```
        Specifications for component subgraphs
  (Anthropological Linguistics 20:167-183, 1978)?
                  Y for yes, N for no:
```

asks whether the full information about cooccurrence and lack of cooccurrence should be displayed in the form of a set of subgraph specifications that can be combined into a master graph that shows all cooccurrence restrictions in the system.

How to start

If the computer has just been turned on, it will go through a series of startup operations that are described in its manual, and will stop after asking you for the date. There will be a period at the left margin written by the computer (and therefore underlined here) just before a spot called the cursor, which blinks on and off. Type

```
.DATE 10-MAR-82
```

or whatever the correct date is, using the form given: one or two digits for the day, hyphen, three letters for the month, hyphen, and two digits for the year. End the line by pressing the RETURN key. Then set the time on a 24-hour clock scale by typing a line of the form

```
.TIME 14:25:40
```

and RETURN. Now you are ready to call up PTP. If it is on the same tape or disk as the RT-11 system itself the callup is

```
.R PTP
```

followed by the RETURN key (with which all lines are ended). If it is on some other device, see the RT-11 manual for the correct way to start a program from that device using RUN and a device designation prefixed to PTP.

As soon as PTP is ready to go the screen says

```
PROGRAMMABLE TEXT PROCESSOR VERSION XX IMM 1:
```

This tells you that PTP is ready, gives you the version of PTP you are using in case anything should go wrong--nothing will--and tells you that you are in the immediate mode (IMM) of PTP, the mode in which it takes directions from the keyboard rather than from memory, and that you are in Buffer 1 where PTP programs always go.

If the computer is already running in RT-11 and the period is the last thing written on the screen, type

```
.DATE
```

and the screen will show the date the last person set, and

```
.TIME
```

to show the time set by the last person. You can correct either by giving the command followed by a value just as you do on startup. Then type

```
.R PTP
```

and continue from there. As before, each line is ended with the RETURN key.

If there is no period next to the cursor, either the last person failed to terminate his or her work properly, or--once in a blue moon--something may have gone wrong with the computer, which is normally as reliable as your pocket knife. Usually you can terminate a program that is hung up by typing CONTROL/C (hold down the CONTROL key on the keyboard and while holding it, press C) two times in succession. If nothing happens you may have to go the hardest route by turning off the computer, waiting fifteen seconds (longer if turning off the computer turns off a disk drive; see your hardware manager), turning it on, and starting all over again.

If the last person left PTP running for you, the top of the screen should show IMM for immediate mode followed by a buffer number 1 to 16. Under it there will be a diamond shaped PTP cursor (<>) in addition to the blinking cursor built into the computer. If you are in PTP immediate mode, press the B key to get to Buffer 1. If there is anything on the screen besides the IMM 1: at the top, type 2000w as many times as you need to to clear the screen. Then you are ready to load PARADIGM.

To load PARADIGM, or any other PTP program, make sure you are in PTP immediate mode in Buffer 1 with a clear screen. On a VT-103 computer with tape cartridges the system tape, including RT-11 and PTP, should be in the left-hand tape slot and the PARA program either on the system tape or on a tape that is mounted in the right-hand tape slot. (The computer's own version of the program name gets compressed to PARA because RT-11 allows no more than six characters in a program name.) Type

```
cccl     Input from? sy:para
```

if the program is on the system tape SY:, or else type

```
cccl     Input from? para
```

if the program is on a tape in the right-hand slot. This last form is also the one to use if you are on a disk system.

"cl" is the PTP command for "cartridge load." The "cc" before it is for safety's sake; it means "cartridge close" and makes sure that the channel over which the information is to be read is available. As soon as "cc" is typed it disappears from the screen. The cartridge load command asks for the name of the file to be read, which in this case is

PARA or para

as you like. As always, a RETURN ends the line.

When the program is loaded in, PTP returns to immediate mode with IMM 1: on the top line and the diamond cursor (<>) below it, but now the screen is full of coded letters and numbers. These are PTP commands to be executed in program mode, which takes them up one by one out of memory instead of getting them from the keyboard as immediate mode does. They are the instructions that make the computer behave the way this manual says it should; if you like, they are PARADIGM. Once those commands are loaded, you start the computer through them by typing xx, the command to execute.

When you finish you should do one of two things. Either run PARADIGM again by typing bxx as before, or return from PTP back to the operating system. To do this type a period. At the top of the screen the message

```
RETURN TO OPERATING SYSTEM (Y/N)?
```

appears. You type a Y, upper or lower case, to return to the operating system, or an N, upper or lower case, to remain in the PTP system. Don't type Y until you have made sure that the system tape is back in the left-hand slot if you took it out.

If you do return to the operating system, the characteristic period eventually appears at the left of the screen, ready for another RT-11 operation or for the next person to use the machine.

7 Test Data

The program package for PARADIGM currently consists of nine files:

Documentation

PARABK.MS	or PARAMS.MS (use one or the other, but not both, in the same printout)
PARA01.MS	to be run under MS, the Manuscripter, using ACCENT.CCT as the table of consistent changes
PARA02.MS	
PARA03.MS	
PARA04.MS	
PARA05.MS	
PARA06.MS	
ACCENT.CCT	consistent change table

Program

PARADI.PTP	the fully commented runnable version
PARA.PTP	the faster version compressed from PARADI.PTP

Test data

HCH.DAT	typical correct data from Huichol of Mexico
HCHSEM.DAT	semantic counterpart of HCH.DAT, including hidden sets
MAPU.DAT	correct data with hidden sets from Mapudungu of Chile

TCVERB.DAT data with order cycles from Tucano
of Colombia

Typical correct data

Typical correct data that have been run are the Huichol verb prefixes given as the example in chapters 1 and 4. They are in a file named HCH.DAT. The results are given in those chapters, along with the reference for the total analysis.

```
p&            ka1-ka2-ni
p&-ka2        m&-ni
ka1-p&        m&-ka2-ni
ka1-p&-ka2        (blank line)
m&            ka2
m&-ka2        ke
ni            ke-ni
ka1-ni
```

Data with hidden sets

Data that are correct and are perfectly analyzable, but in which multiple intersections mask some of the distinct complete sets or cliques, are exemplified by a small selection from Mapudungu of south central Chile. These were verified for me by Robert Croese. The semicolons in the data prevent the latter part of the string from being analyzed in a test run, since those suffixes are not involved in multiple intersections. The data are in a file named MAPU.DAT.

```
(amu)a-l;m-iS       ;if you go -- future  (MAPUDUNGU)
(amu)la-a;I-m-iS    ;you will not go
(amu)no-a-l;m-iS    ;if you do not go
(amu)lle-a;I        ;he will definitely go
(amu)lle-l;m-iS     ;if you really go
(amu)chi            ;let me go
(amu)ki...l-chi     ;don't let me go
(amu)ki...l-no-chi; ki-no-l don't let me go
(amu)lle-chi        ;let me go really
(amu)lle-no-l;m-iS;if you really do not go
(amu)lle-la;I       ;he definitely did not go
```

Another set of data are also from Huichol; they are the semantic features that arise as a result of the analysis of HCH.DAT. The full discussion of how one goes from the primary data on forms to the semantic analysis is discussed in my "How to Recognize Systems" in Greaves and Benson, eds., *Systemic Perspectives on Discourse*: Theoretical Papers, to be published by Ablex. The test data in the file HCHSEM.DAT give the results of that analysis. They are complexes of features rather than strings of forms, so positional analysis does not apply to them. In the analysis of distinct sets, however, they illustrate hidden sets.

pos-asr	;positive-assertive HUICHOL
asr-neg	;negative
pos-mod-asr	;moderated
mod-asr-neg	;"perhaps"
pos-dep	;dependent
dep-neg	
pos-nar	;narrative
pos-mod-nar	;(= narrative alone)
neg-nar	;(requires moderated)
pos-evl	;evaluative (idiom)
evl-neg	
pos-cnj	;conjunct
neg-cnj	
pos-imp	;imperative
pos-imp-dir	;direct (idiom)

Order cycles

Data that contain cycles always cause the positional analysis process to fail. One set of test data with such cycles has been developed from work on Tucanoan verb suffixes compiled in Colombia by Terry Malone. They are in a file named TCVERB.DAT. "vbs" stands for "verb stem", and the three-letter codes at the end of each form are abbreviations for the category labels "past, present, future, imperative, interrogative" rather than the actual forms, whose variants require more disentangling than is worth doing here. The letter + is a high central vowel, the apostrophe ' is the glottal stop, and the tilde ~ is nasalization.

vbs-al-sĩ'rĩ-ti-pas
vbs-al-sĩ'rĩ-pas
vbs-bo-int
vbs-bosa-cũ-fut

vbs-m+'tã-fut
vbs-m+'tã-sĩ'rĩ-pre
vbs-m+jã-fut
vbs-m+jã-mi-pas

vbs-bosa-fut
vbs-bosa-imp
vbs-bosa-int
vbs-bosa-n+'cã-pas
vbs-bosa-nu'cũ-cã'-pas
vbs-bosa-nu'cũ-cã'-pre
vbs-bosa-sĩ'rĩ-pre
vbs-butia-pre
vbs-cusija-cã'-pre
vbs-cusija-fut
vbs-cã'-cũ-pas
vbs-cã'-mi-pas
vbs-cã'l-no'-pre
vbs-du'u-cã'-fut
vbs-du'u-cã'-pas
vbs-du'u-imp
vbs-du'u-ti-cã'-imp
vbs-ja-no'-pre
vbs-no'-sĩ'rĩ-pre
vbs-no'-ti-cã'-imp
vbs-nu'cũ'-cã'-sĩ'rĩ-pas
vbs-nu'cũ-cã'-fut
vbs-nu'cũ-cã'-imp
vbs-nu'cũ-cã'-pas
vbs-nu'cũ-fut
vbs-nu'cũ-imp
vbs-nu'cũ-sĩ'rĩ-pre
vbs-ña'-cã'-pre
vbs-ña'-fut
vbs-ña'-imp
vbs-pe'ti-cã'-pas
vbs-pe'ti-toja-pas
vbs-po-imp
vbs-que'ti-ti-cã'-imp
vbs-seti-a-pas
vbs-seti-b+ro-pas
vbs-seti-mi-int
vbs-seti-nu'cũ-pas
vbs-seti-t+'sa-pas
vbs-sĩ'rĩ-fut
vbs-sĩ'rĩ-imp
vbs-sĩ'rĩ-int
vbs-sĩ'rĩ-mi-pas
vbs-sĩ'rĩ-pre
vbs-sĩ'rĩ-ti-fut
vbs-sĩ'rĩ-ti-mi-pas
vbs-sĩ'rĩ-ti-pas
vbs-sĩ'rĩ-ti-pre
vbs-t+'sa-fut
vbs-t+'sa-y+'r+a-pas
vbs-tamo-sĩ'rĩ-pas
vbs-ti-bo-pas

vbs-masĩ-imp
vbs-masĩ-ti-pas
vbs-masĩ-ti-pre
vbs-mi-int
vbs-n+'cã-bo-pre
vbs-n+'cã-imp
vbs-n+'cã-masĩ-ti-pas
vbs-n+'cã-ti-pas
vbs-n+'cã-y+'r+a-pre
vbs-n+'cõ-sĩ'rĩ-pre
vbs-nemo-imp
vbs-nemo-sĩ'rĩ-pre
vbs-nemo-ti-cã'-imp
vbs-nemo-ti-pas
vbs-nemo-wã'cã-pas
vbs-no'-fut
vbs-no'-imp
vbs-no'-m+jã-pas
vbs-ti-bo-pre
vbs-ti-butia-int
vbs-ti-cã'-imp
vbs-ti-fut
vbs-ti-pas
vbs-til-cũ-pas
vbs-til-fut
vbs-turia-imp
vbs-wã'cã-fut
vbs-wã'cã-imp
vbs-y+'r+a-imp
vbs-y+'r+a-int
vbs-y+'r+a-wã'cã-pas
vbs-yu-int
vbs-yu-m+'tã-pas
vbs-yu-mi-pas
vbs-yu-pas
vbs-yu-toja-pas
vbs-yu-toja-pre
vbs-y+'r+a-wã'cã-pas
vbsl-cusijal-vbs-cã'-pas
vbsl-mil-toja-pas
vbsl-n+'cãl-vbs-pas
vbsl-nu'cũl-vbs-pas
vbstr-cã'-sĩ'rĩ-mi-pre
vbstr-masĩ-pe'o-imp
vbstr-n+'cõ-pe'o-cã'-imp
vbstr-n+'cõ-pe'o-imp
vbstr-pe'o-cã'-fut
vbstr-pe'o-cã'-pas
vbstr-pe'o-fut
vbstr-sĩ'rĩ-ti-cũ-pas
vbstr-wã'cõ-n+'cõ-pas

8 Program Listing

The rest of this book is a listing of the actual PARADIGM computer program. There are two versions: PARADI.PTP, the heavily commented runnable version reproduced in this chapter, and PARA.PTP, the compressed version derived from PARADI.PTP by a utility program. PARA.PTP runs considerably faster, and has more space for data vectors and matrices, but it is nearly incomprehensible to anyone but a computer.

PARADI.PTP, on the other hand, should offer little difficulty to any experienced programmer even if he has no previous knowledge of the PTP language in which it is written. This is because the ratio of comments to actual program statements is on the order of 8 to 1. Each line of the program begins with an explanation in a kind of restricted English of what the code on that line is intended to accomplish. The code follows at the end of the line. Subroutines are prefaced with an imperative statement that characterizes what the subroutine as a whole is supposed to do.

The PTP code as written holds to the basic programming constructs of sequential, repeated, and alternative execution. The sequential code is recognized as statements whose comments have the form

```
"Do ...." xyz
"And ..." xyz
"And ..." xyz
```

where the xyz represent short sequences of PTP one-letter codes. The details of these are given in the PTP programmer's reference manual by Simons and Woods that was referred to in chapter 6 (which also gives information needed for making and testing modifications in PTP programs).

The repeated code permits the most general form of loop structure which is sometimes called the "one and a half times" loop:

```
("REPEAT: do this ..." xyz
 "UNTIL this condition is satisfied" xyz?;
 "MEANWHILE do this other thing and loop" xyz:)
```

Making either the part before the test or the part after the test null yields the familiar "test at the beginning" and "test at the end" constructs, for which the form of the PTP comments is

```
("WHILE this condition is not satisfied" xyz?;
      "Do this ..." xyz:)
```

or the equivalent and more convenient

```
("WHILE this condition is satisfied" xyz?
      "Do this ..." xyz:)
```

and

```
("REPEAT: do this ..." xyz
 "UNTIL this condition is satisfied" xyz?;:)
```

Alternatives match a standard IF-THEN-ELSE syntax with the capability of extension by ELSE IF conditions:

```
("IF this condition is satisfied" xyz?
 "THEN do this" xyz;
 "ELSE IF another condition is satisfied" xyz?
 "THEN do this other" xyz;
 "ELSE do the default alternative" xyz)
```

Subroutines are defined in the form

```
@s"Statement of intent (called from t)"
("Body of the subroutine ..." xyz)
#
```

Within many subroutines the code is divided into up to three conceptually useful parts:

SETUP
PROCESS
CLEANUP

The stated intent of the subroutine is carried out in the PROCESS part. The SETUP and CLEANUP parts get things into place and put them back into place. These three parts do not, technically speaking, always divide a subroutine into its logical subtrees; instead they serve the human reader to help him pinpoint where, so to speak, the action is.

The subroutines of which the program principally consists are listed in alphabetical order. The root of the program is the one-line, one-statement call

xC

in the first line of the program, which activates all the rest when the immediate execution command *xx* is typed on the keyboard. Calls to a subroutine consist of the code *xs*, where *s* is the one-letter identifier that appears immediately after the *@* sign in the definition.

Changes to this program that are of general interest, or fixes to bugs in it, or requests to fix bugs, or proposals for other implementations, should be communicated to the author:

Joseph E. Grimes
PARADIGM Project
Language Data Processing
International Linguistics Center
7500 West Camp Wisdom Road
Dallas TX 75236 USA

This program was written entirely in the PTP language. The three algorithms in the main procedure would run faster in a language like C or Pascal or APL or PL-1 that handles bit arrays more efficiently; but at the time the work was done, using VT-103 computers at the Summer Institute of Linguistics field stations in Peru and Colombia, none of those programming languages was readily available and PTP was. On the other hand, PTP is one of the best languages there is for reading in the data and transforming them to a form that can be processed. All things considered, a program that works right is more useful, even if parts of it are a little slow, than one that doesn't do the job or that never gets written. Now that the design is proven, a more efficient implementation might be taken up.

```
xC#"/Type xx to start/"
@02cqPARADIGM.PTP Version 2.209^M^J$#


"Positional and cooccurrence analysis"
"Joseph E. Grimes, 78 June 1 to 79 Jan 6"

 Positions J. Albert Bickford 79 Nov 15
                                   to 80 Sep 5"
"Redesigned JEG 82 Feb 10 to 82 Jul 28"

"Channels"
" 1 printer"
" 2 screen"
" 3 input tape"
" 4 list of morphemes to analyze"

"Buffers"
" 8  blank data vector"
" 9  page heading"
" 10 morphemes"
" 14 precedence matrix, union vectors"
" 15 data vectors"
" 16 scratch buffer"

"Variables"
" 1  number of morphemes in B10 list"
" 2  morpheme index, L"
" 3  morpheme length, L; row sum, W;
                            column sum, F"
" 4  index of preceding morpheme, L;
                     lowest nonzero sum, F"
" 5  local use"
" 6  local use"
" 7  padding width"

" 10 count of unassigned rows"
" 11 current position class"
" 12 1= class assignment made, 0= no"

" 18 column width, 12 or more"
" 19 first tab stop, initially 5"
" 20 characters left in line, d"
" 21 lines left on page"
" 22 characters per line (55)"
" 23 lines per page (60)"
" 24 line break zone (10)"
" 28 1= print matrix, 0= no"
" 29 1= first page printed, 0= not yet"

" 30 1= analyze all morphemes, 0= some"
" 31 1= complete printout including data,
                                  0= no"
" 33 1= position analysis, 0= no"
" 35 1= distinct sets of mutually
                  exclusive forms, 0= no"
" 36 1= subgraphs, 0= no"
" 37 1= monitor processes, 0= no"
" 38 1= hidden set found, 0= no"
" 39 number of reductions"

" 40 buffer index, B; variable index, i;
                             buffer to sort, S"
" 45 number of position classes, Q"
" 48 1= printer, 2= tape"
" 49 1= positional ambiguity, 0= none"
" 50 1= terminate, 0= continue"
#

@C"MAIN PROCEDURE"
"  SETUP"
("Initialize"xi
"Read in forms"xf

"  PROCESS"
"Positional analysis"xo
"Distinct sets"xm
"Component subgraphs"xs

"  CLEANUP"
("IF no abnormal termination"b50pz?
 "THEN"2cqAll requests completed.^M^J$;
 "ELSE"2cqRun terminated abnormally.^M^J$)
"Beep"2cq^G$
"End output page"cq^L$
"Close channels"1cc2cc3cc)
#

"SUBROUTINES FOLLOW IN ALPHABETICAL ORDER"

@A"Change list to one morpheme per line
                                (from I,M,r)"
("Beginning of line" u
("REPEAT:"
       [("IF left parenthesis" pi($/)$a?
            "AND right parenthesis" gpi($
                                       ps)$?
         "THEN delete in between" \

             ("IF noninitial" pb'?
                 "AND not preceded by
                          hyphen" jpi-$k'?
              "THEN put in hyphen" /-$)

              "Replace" /STEM-$

             ("IF followed by hyphen" pi-$?
              "THEN delete it" s;

              "ELSE if at end" (pi^M$?;pm')?
              "THEN delete the new one" a)

              "Exit as true" pt;
       "ELSE test next character" k)]

"UNTIL stem found" ?;
      "OR end" pm'?;:)

"Beginning of line"u
("WHILE morphemes remain"pm?
      ("IF separator"pw -^M$?
       "THEN new line"o^M$
           ("WHILE other separators follow"pw -^M$?
                  "Delete them"s:);

       "ELSE IF gloss" (pi;$)?
       "THEN delete the rest of the line" w

           ("WHILE spaces precede" jpi $k?
                  "Delete" a:);

       "ELSE advance"k):)

("IF no final <RETURN>"jpi^M$k'?
 "THEN add one"/^M$))
#

@B"Clear buffers, close channels (from i)"
("Save index"40<
"Start with B7"40v7z
10("WHILE buffers remain"
         "Top of buffer"40^b
         "Clear it"xb
         "Next buffer"+)
"Close channels"1cc2cc3cc>)
```

```
#


@b"Clear a buffer (from B,i,J,m,o,r,s,V,>,<)"
("WHILE lines remain"pm?
      "Clear a reasonable number"20w:)
#

@D"End a line on the screen and the printer"
       "(from f,I,j,K,m,N,o,P,s,Z,<,=,\,~)"
(2cq^M^J$
cq^M^J$
"Left margin, 10 spaces"10(cq $)
"Decrement line count"21v-
("IF end of page"0pe?
 "THEN new page"xP))
#

@d"Print a buffer from BP on (from f,M,r,Z)"
"  SETUP"
("Save character counter"20<

"  PROCESS"
("WHILE lines remain"pm?
      "Print the line" xN
       "Next line"i:)>)
#

@E"Print a form and a line (from f,Z)"
"  SETUP"
("Top of matrix" 1^uu
 "Auxiliary on names" [gl0bt

"  PROCESS"
("WHILE lines remain" pm?
      "Set local padding count" 6v7^z
      "Get name list" t

     ("WHILE no end" pi^M$'?
           "Print" cp2cp
           "Count down" -k:)

      "Add padding" 6^(cq $)6^(2cq $)

      "Back to matrix" t
      "Print line" xN
      "Next line and morpheme" itit:)])
#
```

```
@F"Find lowest column sum over data vectors (from s)"
"  SETUP"
("Save index and weight variables"2<3<5<
"Column index"2v0z
"Column with lowest nonzero sum and its
                               weight"4v0z5v5000z
"Column sums"14b

"  PROCESS"
("WHILE column sums remain"pm?
       "Get the sum in V3"3v4n

      ("IF not zero"0pe'?
       "THEN"
           ("IF the sum is least"5v3pu?
            "THEN reset index of lowest n
                                  onzero sum"4v2^z
                 "Reset lowest nonzero va
                                  lue"5v3^z

                ("IF monitoring"37pz'?
                 "THEN"(2cqLowest column
                                  sum$)2cw2cq^M^J$)))

       "Next sum"2v+i:)>>>)
#

@f"Read in forms (from C)"
("IF continuing"50pz?
"THEN"

    ("IF printout wanted"31pz'?
     "THEN start page"xP
           cqData to be analyzed$2(xD))

     "Distinguish case"pd
    ("WHILE data lines remain"xr?
           "Look up morphemes and build tables"xL:)

    ("IF no data"1pz?
     "THEN say so"cqNo data to be analyzed$xD
          "Set the termination switch"50v1z;

     "ELSE sort the data vectors" 40v15zxS
          "Padding counter" 7v0z
          "On morpheme list" 10b

         ("WHILE morphemes remain" pm?
                "Local variable" 5v0z

               ("WHILE characters remain"
                                        pi^M$'?
                         "Count and advance" +k:)

                  ("IF largest" 7pu?
                   "THEN update padding" 7v5^z)

                   "Next morpheme" i:)

            "Add 3 to padding" 7v3+)

     ("IF printout wanted"31pz'?
         "and no termination"50pz?
      "THEN print the morpheme list"xM)

     ("IF matrices are to be printed" 28pz'?
      "THEN"xDcqData vectors$2(xD)
           "Data vectors"15bxd3(xD)

           ("IF positional analysis" 33pz'?
            "THEN"cqPrecedence matrix for
                     positional analysis$2(xD)
            "Print" 14bxE3(xD))))
#

@G"Put morphemes for the vector at SAVE
                              into B16 (from k)"
"  SETUP"
("Auxiliary on morpheme list"[10bg
"Bottom of B16"16bps%%$
"Back to vector"e

"  PROCESS"
1^("WHILE positions remain in the vector"


        ("IF there is a morpheme"pil$?
         "THEN get it from the morpheme list"trt
              "Change its <RETURN> to
                                 <SPACE>"ejo $e)

         "Next morpheme in list"tit
         "Next position"k)

"  CLEANUP"
"End of line"e
("WHILE spaces remain"jpi $k?
       "Trim"a:)
"Final <RETURN>"/^M$
"Set save back on the next vector"ed])
#
```

```
@g"Get a character string from the keyboard (from
I,i)"
("REPEAT: get one character"1j

        ("IF <DELETE>"pi$?
         "THEN erase it and the one before"sa)

 "UNTIL <RETURN> (erase it when found)"
                                 pi^M$?s;k:)
#

@H"Open an output device (from i)"
("Ignore case"pq
("REPEAT"2cqOutput to printer or tape (P or T)? $
 "UNTIL acceptable"1j2cq^M^J$pwPT$?;s:)

("IF printer"48vpiP$s?
 "THEN"1z2cqTurn the printer on,
           then press <RETURN>.$
      "Wait"1a2cq^M^J$
      "Open the printer"(coLP:$);
 "ELSE"2z2cqTape for output in place,
                            then press <RETURN>. $
      "Wait"1a2cq^M^J$
      "Channel 1"co$))
#

@I"Find what morphemes to analyze (from i)"
("IF"2cqAnalyze all morphemes in the data? $30vxy?
 "THEN V30 = 1 and B8 is empty"8b/^M$;

 "ELSE"
     ("IF"2cqIs the list of morphemes to
             analyze in a file?^M^J$2vxy?
      "THEN read them into B10"10b4cl$;

      "ELSE get them from the keyboard"
            2cq    Enter the morphemes to
                   analyze, separated by spaces^M^J$
            2cq      and followed by <RETURN>^M^J$
           "Get the list"10bxg)

      "Upper and lower case different in data"pd
      "Put each morpheme on a line by itself"10bxA
      "Morpheme counter is V1 throughout program"v
      "Top of list"(10upb?;:)

     ("IF the list is not empty"pi^M$'?
      "THEN"
          ("WHILE morphemes remain"pm?
                 "Look at the following morphemes"gi
```

```
               ("WHILE others remain"pm?

                    ("IF a duplicate"p1?
                     "THEN eliminate it"w;
                     "ELSE next morpheme"i):)

                "Next morpheme to compare"ti+:))

     ("IF there were elements in the list"pz'?
      "THEN set up a blank array for them"
                                8b^(/0$)/^M$
          "Blank precedence matrix in B14"
                                ug14b^(y);

      "ELSE"xD(cqNo morphemes given in the
                  list to analyze.^M^J$
              cqAll will be processed.^M^J$)
          "Reset the switch for all morphemes"30v1z
          "Clear the list buffer"8bxb/^M$))
#

@i"Initialize (from C)"
"  SETUP"
("Clear buffers"xB
"Open the screen"2cott:$
"Program name on screen"x0
"Morpheme counter zero"1v0z
"Termination switch off"50v0z

"Characters per line"22v55z
"Lines per page"23v60z
"Line break"24v10z
"First page switch"29v0z
"Initial tab stop"19v5z

"Ignore case"pq

"  PROCESS"
"Open the output device"xH

("IF printing"48v1pe?
 "THEN clear heading buffer"9bxbm
      (2cqType the page heading,
          then press <RETURN>.^M^J$
       2cqTITLE          DATE          TIME^M^J$)
      "Get the heading from the keyboard"xg
      "Add a page number"/   000$)

 "Find what morphemes to analyze" xI
 "Find what processes to use" xJ)
#
```

```
@J"Find what processes to use (from i)"
("IF termination is not set"50pz?
 "THEN display empty buffer"16bxbm
       2cqTape for input in right hand slot,
          then press <RETURN>.$1a2cq^M^J$
      (2cqData source file:^M^J$)
      "Open the input file"3ci$

      "Ignore case"pq

      "Ask about processing options"
       2cqComplete processing? $
     ("IF complete printout wanted"31vxy?

      "THEN set all switches on"40v32z5
             (40^v1z40v+);

      "ELSE ask about each"2cqAre the input data
                               to be printed? $31vxy
            2cqPositional analysis
           (Language 43:437-444, 1967)?^M^J   $33vxy
            2cqDistinct sets of mutually
                             exclusive forms? $35vxy
            2cqSpecifications for
               component subgraphs^M^J$
            2cq  (Anthropological Linguistics
                     20:167-183, 1978)?^M^J   $36vxy)

        2cq^JMonitor the processes on the screen?^M$
        2cq^J(WARNING: slow) $37vxy

        2cqPrint the internal matrix
                             representations? $28vxy)
 #

 @j"Specifications for subgraph from column in V4
                                           (from s)"
 "  SETUP"
 ("Extra line"xD
  "Clear work buffer" 16b(pm?w:)

  "Auxiliary on morpheme"10b4^ig
  "Print its name" cq  SUBGRAPH FOR $xN
  "Go back" tg

  "First data vector"15b

 "  PROCESS"
 ("WHILE data vectors remain"pm?
        "Save the place"d
      ("IF the morpheme is in the vector"
                                   4^kpil$?
       "THEN beginning of vector"ed

           ("IF monitoring"37pz'?
            "THEN"(2cqSubgraph: $)2cw)

            "Remove the row counter"ps $a3su
            "Get the morpheme shapes" xk

            "In B16" [gl6b
            "Print the line of shapes" xN

            "Erase it" uw
            "Back to the row" t]

            "Delete that vector"qs;

       "ELSE skip it"i):))
#

@K"Test for zero vector and print it (from s)"
("IF data vector is blank"ps $pi000$u?
 "THEN print it"xD(cq---$)xD
"  CLEANUP"
      "Erase it"w
      "Exit as true"pt)
#

@k"Print the morpheme in B10 that has auxiliary"
"and its associated vector that has save (from j, Z)"
"  SETUP"
("Test the vector" e
 "Switch off" 5v0z
"  PROCESS"
("WHILE still off" 0pe?
       "AND not at end" pi^M$'?

      ("IF a set member" pil$?
       "THEN switch on" lz)

       "Next component" k:)

 "Back to earlier place" kue

("IF data present" lpe?
 "THEN get morphemes that match the vector"xG;
 "ELSE go back" e))
#
```

```
@L"Look up morphemes"
"Then set precedence and data vectors (from f)"
"  SETUP"
("Add a blank data vector"15bxz
"Auxiliary on it"ug
"Set index for preceding element null"4v0z

"  PROCESS"
("IF there are morphemes in the data line"16bpi^M$'?
 "THEN"
     ("WHILE morphemes remain in it"pm?
            "Auxiliary and save on the morpheme
                                        in B16"[gd

           ("IF this morpheme (index in V2)
                                      is new"xn?
            "THEN"
                ("IF all morphemes are included"
                                          30pz'?
                 "THEN add it to the list"xppt)'))

           ("IF old or just added"'?
            "THEN set it in the data vector"
                                      tku2^kjol$ut

                ("IF anything precedes it"4pz'?
                 "THEN set the precedence matrix"14b
                      "Row of preceding morpheme"
                                            4^iu
                      "Column of this one"2^kjol$)

                 "Reset the preceding element"4v2^z)

            "Next morpheme"ei:))

     ("IF monitoring"37pz'?
      "THEN"(2cq   $tku2cwt)))
#

@M"Print the morpheme list (from f)"
"  SETUP"
("Use V5"5<
"Morpheme list"10b
"Number of morphemes in V5 less one"5v^z-

"Change returns to spaces"5^(ijo $)

"  PROCESS"
"Header"3(xD)cqMorphemes to be analyzed$
"Set up a counter"11bxb/ (000)$2jl^&u
"Print the count"cww2(xD)
```

```
"Print the list"10bxd2(xD)
"  CLEANUP"
"Restore the list to one element per line"10bxA>)
#

@m"Distinct sets (from C)"
"  SETUP"
("IF continuing"50pz?
    "and requested"35pz'?

 "THEN clear the scratch buffer"16bxbm
       2cqDistinct sets of mutually
                       exclusive forms^M^J$
      "New page"xP
       cqDISTINCT SETS OF MUTUALLY
                       EXCLUSIVE FORMS$2(xD)

"  PROCESS"
      "Fill the union matrix"xV
      "Top of matrix"1^uu
      "Row index" 2v0z

     ("WHILE rows remain"pm?
            "Set identity"2^kol$
            "Advance index and pointer"+i:)

      "Find and print the sets"xZ

"  CLEANUP"
("WHILE there are any hidden sets left" 38pz'?
       "Blank line" xD
       "Form a reduced matrix" xR
       "Find and print the sets" xZ:))
#

@N"Print a line (from d,E,j)"
("Reset character count"20v22^z

("WHILE not end of line"(pi^M$;
       "OR end of buffer" pm')'?

      ("IF monitoring" 37pz'?
       "THEN show" 2cp)

       "Print and advance" cpk
       "Count down"20v-

      ("IF in the end zone"24po?
          "AND on a separator"pw -$?
       "THEN"

           ("IF monitoring" 37pz'?
            "THEN show" 2cp)

            "Print and advance" cpk
            "Break the line"x~;

       "ELSE IF end of line"20v0pe?
       "THEN"

           ("IF monitoring" 37pz'?
            "THEN show break" 2cq =$)

            "Tag and break" cq =$x~):)

"Print line end"xD)
#

@n"Test if a morpheme is in the list (from L)"
"  SETUP"
("Initialize V2 to 1"2v1z
"Morpheme list"10b

"  PROCESS"
("REPEAT UNTIL end of list (true)"pm'?;
        "or match (false)"pl?';
        "Next morpheme"i+:))
#

@o"Positional analysis (from C)"
"  SETUP"
("IF continuing"50pz?
    "and requested"33pz'?
 "THEN clear the scratch buffer"16bxb
       2cqPosition classes^M^J$
      "New page"xP
       cqPOSITION CLASSES$xD
       cq  Predecessor and successor
                           numbering is arbitrary.$xD
       cq  Combine the two to identify spans, then$xD
       cq   number the relative orders
                           out from the stem.$2(xD)
       cq    ... -3  -2  -1   |  0    |
                                  1   2   3 ...$xD
       cq    -----------------+-------+
                               ----------------$xD
       cq        Prefixes     | STEM  |
                                       Suffixes$xD

      "Precedence matrix"14b

"  PROCESS"
      "Analyze predecessors"x>
      "Analyze successors"x<)
```

```
#

@P"New page (from D,f,m,o,s)"
(("IF first page not started yet"29v0pe?
 "THEN set its switch"lz;

 "ELSE IF not top of page" 21v23pv'?
 "THEN form feed"cq^L$)

"Three blank lines at top"3(cq^M^J$)
"Margin" 10(cq $)
"Reset the line counter"21v23^z

"Save the place"[g
"Increment page number in heading"9bij&u
"Print the heading"cw2(xD)
"Back"t])
#

@p"Add a new morpheme (from L)"
("Put it on the morpheme list"y
"Extend the blank vector"8b/0$

"In the precedence matrix"14b
1^("WHILE rows remain"
         "Add zero to the end"ij/0$k)
"Add a blank row at the end"xz

"In the data vectors"15b
("WHILE vectors remain"ps^M$?
       "Add zero to the end"j/0$k:)

"Increase the morpheme count"v+)
#

@R"Form a reduced matrix (from m)"
"  SETUP"
("Top of matrix" 14b
 "Reduction count at zero" 39v0z

"  PROCESS"
("WHILE rows remain" pm?

      ("IF the row contains a hidden set" pi3$?
       "THEN leave it in" i;

       "ELSE IF no hidden set in the row" pi^M$?
       "THEN beginning of row" u

           ("WHILE positions remain" pi^M$'?
                  "Change to minus" o-$:)
```

```
            "Reduction count" +
            "Next row" i;

       "ELSE advance" k):)

"  CLEANUP"
"Back to the top" 1^uu
("IF hidden sets remain" 38pz'?
 "THEN"

     ("IF no reduction" 39pz?
      "THEN show error condition" m
            2cqHidden sets cannot be eliminated.
               <SPACE> to continue.$
           "Wait" la;
      "ELSE eliminate columns" xT)))
#

@r"Read one line of data (from f)"
"  SETUP"
("Clear a scratch buffer"16bxb
 "Left end of line"=2cqData : $

"  PROCESS"
("IF a line cannot be read"3cr'?
 "THEN exit as false"';

 "ELSE back to beginning"u

     ("IF it is to be printed"31pz'?
      "THEN print it"xd)

      "Display it"2cw

      "Break it up into one morpheme per line"xA
      "Exit as true"pt))
#

@S"Sort the buffer whose number is in V40 (from f,Z)"
"Major lines begin with a nonblank character"
"Minor lines begin with blank and follow major lines"
"  SETUP"
("Top of the buffer to sort"[40^b
 "Second major line"(ipi $'?;:)

"  PROCESS"
("REPEAT: auxiliary on the new line"
          gijl4mu(2cq  Sorting...$)
         "Compare with the lines above"(20upb?;:)

        ("WHILE lines are minor"(pi $?;
               "OR too large"tpgt)?
               "Advance"i:)

        ("IF auxiliary"px?
         "THEN no change, so next line"(ipi $'?;:);

         "ELSE set counter"5vlz
              "Set auxiliary"[g
             ("WHILE minor lines follow"ipi $?
                    "Increase the count"+:)
              "Back to major line"t]

             ("IF full duplicate"5^pl?
              "THEN erase"t5^wt;

              "ELSE insert lines"5^y
                   "Erase where they came from"
                                           t5^wt))
 "UNTIL end"pm'?;:)

 "Back"(20upb?;:)m])
#

@s"Component subgraph specifications (from C)"
"  SETUP"
("IF continuing"50pz?
    "and subgraphs requested"36pz'?
 "THEN V4 for column index"4<
       "New page"xP
       cqSPECIFICATIONS FOR COMPONENT SUBGRAPHS$2(xD)

      "Clear the scratch buffer"16bxb
      "Clear B14 for column sums"14bxb
      "Set blank column sums"1^(/0000^M$)


      "Go through data vectors"15b
     ("WHILE data vectors remain"pm?
            "Add a row counter with auxiliary on it"
                                           ij/ 000$jg
            "Beginning of vector"u

         1^("WHILE positions remain"
                 ("IF one"pil$?
                  "THEN count"t&t)
                  "Next position"k)

           ("IF monitoring"37pz'?
            "THEN"(2cqRow sums: $)ku2cw)

            "Next vector"i:)

"  PROCESS"
```

```
     ("WHILE data vectors remain"15bpm?

           ("IF only one vector is left"ipmu'?
            "THEN see if it is all zeros" xK)

           ("IF nonzero vectors remain"'?

            "THEN count the weight of each column"xW
                 "Find the lowest column sum"xF
                 "Print specs for that column"15bxj
                 "Blank line"xD)

            "Next pass":)>)
#

@T"Eliminate columns from a reduced matrix (from R)"
"  SETUP"
("Column index" 5v0z

"  PROCESS"
("WHILE not at end" 1^uu5^kpi^M$5^j'?
       "Local hidden set switch off" 6v0z


       ("WHILE rows remain" pm?

             ("IF this column contains a hidden set"
                                            5^kpi3$?
              "THEN hidden set switch on" 6v1z
                   "Jump to the end" 1^ii;

              "ELSE next row" i):)

       ("IF no hidden set in the column" 6pz?
        "THEN top of buffer" 1^uu

            ("WHILE rows remain" pm?

                   "Make cell a minus" 5^ko-$
                   "Next row" i:))

        "Index on next column" 5v+:)

"  CLEANUP"
"Top of buffer" 1^uu
("IF monitoring" 37pz'?
 "THEN display" ijmu
       2cqReduce the matrix^M^J$)

("WHILE cells remain" pm?

      ("IF changed" pw23$?
       "THEN change back" ol$;
       "ELSE advance" k):)
"Back to top" 1^uu)
#

@V"Fill the union matrix (from m)"
"  SETUP"
("V2 is position index"2<
 "Clear the precedence array"14bxb
 "Blank union matrix"g8btl^(y)

"  PROCESS"
"Data vectors"15b
("WHILE vectors remain"pm?
       "Zero the position index"2v0z

    1^("WHILE positions remain in the vector"
             ("IF filled"pil$?
              "THEN save the place"[g
                    Union vectors"[gl4b

                  ("IF monitoring"37pz'?
                   "THEN display"ijmu
                         2cqData vectors to union
                                     vectors^M^J$)

                   "Find the union vector
                              for the position"2^i
                   "Back to position"t
                   "Beginning of data vector"ku

                1^("WHILE positions remain"

                        ("IF one"pil$t?
                         "THEN unite"ol$;

                         "ELSE advance"k)

                         "Next position"tk)]

                   "Back to position"t])
              "Next position"+k)
        "Next vector"i:)

"  CLEANUP"
"Complement the union vector"xw>)
#
```

```
@W"Calculate the column sums (from s)"
"  SETUP"
("V3 holds row count"3<3v
  Zero column sums"14bl^(o0000$i)
 "Auxiliary at top"l^uug
 "Data vectors"15b

"  PROCESS"
("WHILE data vectors remain"pm?
       "Get the row count"ps $3n
       "Auxiliary on first column sum"tl^uut
       "Beginning of row"u

    l^("WHILE positions remain"

            ("IF one"pil$?
             "THEN get that column counter"t
                  "Increment it"3k3^&t)

             "Next position"k
             "Next column counter"tit)

       ("IF monitoring"37pz'?
        "THEN display from top"tl^uul4mt
              2cqColumn sums^M^J$)

        "Next data vector"i:)>)
#

@w"Complement the union vector (from V)"
"  SETUP"
("Top of matrix"14b

"  PROCESS"
l^("WHILE rows remain"
      l^("WHILE positions remain"

              ("IF zero"pi0$?
               "THEN one"ol$;
               "ELSE complement to zero"o0$))

        ("IF monitoring"37pz'?
         "THEN display"14m
               2cqComplement of union vector^M^J$)

         "Next row"i))
#
```

```
@Y"Search for hidden distinct sets (from Z)"
"  SETUP"
("Top of matrix" 14b
 "Hidden set switch off" 38v0z
 "Row index" 4v0z

"  PROCESS"
("WHILE rows remain" pm?
       "Index for row" 2v0z

      ("WHILE positions remain" pi^M$'?

            ("IF in an intersection" pi2$?
             "THEN save the place" d
                  "Top of matrix" 4^uu
                  "Symmetric position" 2^i4^k

                 ("IF it is in an intersection too"
                                              pi2$?
                  "THEN it is hidden,
                        so make it a 3" &
                       "Make the other a 3 too" e&
                       "Hidden set switch on" 38v1z;
                  "ELSE back" e))

             "Advance" 2v+k:)

      ("IF monitoring" 37pz'?
       "THEN display" 14m
             2cqFind hidden sets (3)^M^J$)

       "Next row" 4v+i:))
#

@y"Set a yes-or-no option flag (from I,J)"
((2cq Y for yes, N for no: $)
 "Get a one-character answer"1j
 "End the screen line"2cq^J^M$
("IF the answer was Y"piy$s?
 "THEN set the variable designated earlier on
                                    (true)"1z;
 "ELSE off (false)"0z))
#

@Z"Distinct sets of mutually exclusive forms
                                    (from m)"
"  SETUP"
("Use V2, V4"2<4<
"Union matrix" 14b
"Row index"4v0z
```

```
"  PROCESS"
("WHILE rows remain"pm?
       "Index on row beginning"2v0z

      ("WHILE positions remain"pi^M$'?

            ("IF one"pi1$?
               "and not an identity"4^pe'?
             "THEN save place"d
                 "Auxiliary on first position"[kug
                 "Position indicates which row"
                  "to compare with the current row"
                                  u4^u2^i
                 "Main pointer on current row,"
                  "auxiliary on comparison row"t

                ("WHILE positions remain"pi^M$'?
                      ("IF current row has 1"pi1$?
                          "and comparison row
                                     .as 0"tpi0$t?
                       "THEN tag the 1 as"
                            "intersecting with"
                            "another set" o2$j)

                       "Advance"ktkt:)]

                  "Back to current row"e)
             "Next position"k+:)

        ("IF monitoring"37pz'?
         "THEN display"14m
               2cqFind intersections (2)^M^J$)

         "Next row"4v+i:)

"Search for hidden complete sets" xY

"Clear work buffer" 16b(pm?5w:)
"Save pointer on union matrix" 14bd
"Auxiliary on list of morphemes"10bg

1^("WHILE rows remain"
         "Print the subset for that row" xk

         "Next union row"id
         "Next morpheme"tig)

"Sort the output list" 40v16zxS
"Print the subsets" [g16bxdt]
"New lines" 2(xD)

 ("IF matrices are being printed" 28pz'?
 "THEN"xDcqComplemented unions of data vectors
                            -or each morpheme$xD
       cq  with identity matrix added.
                   2=intersecting, 3=hidden$2(xD)
      "Print names and lines" 14bxE3(xD))>>)
#

@z"Add a row of zeros (from L,p)"
("End of this buffer"ps%%$
 "Set the auxiliary there"[g
 "Transfer from the blank row"8bty])
#

@>"Analyze predecessors (from o)"
"  SETUP"
("Zero the position class counter"11v0z
1^("WHILE rows remain"
         "Add a register for class number"/000 $i)
"Class number model with auxiliary on it"
                                   [gllbxb/000$ut

"  PROCESS"
("REPEAT: zero a counter of unassigned rows"10v0z
         "Top of matrix"1^uu
         "Next position class number"11v+

         "Assignment switch off"12v0z

      1^("WHILE rows remain"
              ("IF unassigned"pi000$?
               "THEN count it"10v+
                    "Auxiliary on class number
                               register"[2kg2k

                   ("REPEAT UNTIL end of
                              row (true)"pi^M$?;
                           "or successor
                         found (false)"pil$?';k:)

                   ("IF the row has no successors
                                           in it"?
                    "THEN give it this position
                                      class"tll^&t
                         "Assignment switch on"12v1z

                        ("IF monitoring"37pz'?
                         "THEN display"14m
                               2cqPredecessor class
                                assignment^M^J$))])

                "Next row"i)
 "UNTIL no rows can be assigned (true)"10pz?;
       "or no assignment can be made (false)"12pz?';

       "Set successors in the assigned columns
                                            to 2"xl
       "Print the class"xDcqPREDECESSOR CLASS $x=:)

"  CLEANUP"
"Report inconsistent data if any"x\]
"Top of matrix"1^uu
1^("WHILE rows remain"
         "Clear the class number register"o000$k
      1^("WHILE positions remain"
              ("IF set to 2"pi2$?
               "THEN reset to 1"o1$;
               "ELSE advance"k))i)
"Back to top"1^uu)
#

@<"Successor analysis (from o)"
"  SETUP"
("Extra spaces"2(xD)
 "Zero the position class counter"11v0z
 "Class number model with auxiliary on it"
                                 [gllbxb/000$ut

"  PROCESS"
("REPEAT: zero the counter for unassigned
                                  columns"10v0z
          "Top of matrix"1^uu
          "Next position class number"11v+

          "Assignment switch off"12v0z
          "Zero row index"5v0z

       1^("WHILE rows remain"
                "Set save"d

               ("IF unassigned"pi000$?
                "THEN count it"10v+
                     "Auxiliary on class number
                                   register"[2kg
                      Position index = row + 4"
                                         6v5^z4+
                     "Top of matrix"5^uu

                    ("REPEAT UNTIL bottom
                           of matrix (true)"pm'?;
                            "or predecessor found
                               (false)"6^kpi1$?';
                            "Advance"ji:)
```

```
                   ("IF the column contains no
                                  predecessors"?
                    "THEN give it this position
                                    class"tll^&t
                         "Assignment switch on"l2vlz

                        ("IF monitoring"37pz'?
                         "THEN display"14m
                               2cqSuccessor class
                                assignment^M^J$))])
                "Next row"5v+ei)

 "UNTIL no columns can be assigned (true)"l0pz?;
       "or no assignment can be made (false)"l2pz?';

       "Set predecessors in the assigned rows to 2"
                                                  2
       "Print the class"xDcqSUCCESSOR CLASS $x=:)

"  CLEANUP"
"Report inconsistent data if any"x\])
#

@="Print a position class (from <,>,\)"
"  SETUP"
("Print the number"tcwtxD
 "Top of matrix"l^uu
 "Save at top of morpheme list"dl0be

"  PROCESS"
l^("WHILE rows remain"
        ("IF in the current position class"3pc?
         "THEN get the morpheme"e
              "Hanging indentation"5(cq $)
              "Print up to <RETURN>"
                              (pi^M$'?cpk:)kuxD
              "Back to matrix"e)
         "Next row, both pointers"ieie))
#

@\"Report inconsistent data if any (from <,>)"
("IF no assignment could be made"'?
 "THEN"2cq*** INCONSISTENT DATA ***^M^J$
       2(xD)cqINCONSISTENT DATA$xD
       cqHOMOGRAPHS? LAYERING? ALTERNATE
                                    ORDERS?$xD
       cq  The following morphemes could
               not be ordered successfully$xD

       cq  and are assigned arbitrarily to
                              position class $
       "Set class number model to zero"to000$ut
```

```
      "Print the rest"x=)
#

@~"Break a line being printed (from N)"
("End the line"xD
 "Hanging indentation"19^(cq $)
 "Character count for short line"20v22^z19^-)
#

@1"Set successors in the assigned columns to 2
                                      (from >)"
"  SETUP"
("Top of matrix"1^uu
 "Set class number model"to000$j11^&ut
 "Zero row index"5v0z

"  PROCESS"
1^("WHILE rows remain"
        ("IF in the current position class"3pc?
         "THEN set save"d
              "Position index = row + 4"6v5^z4+
              "Top of matrix"5^uu

           1^("WHILE rows remain"
                   ("IF a one"6^kpil$?
                    "THEN convert to 2"&)i)

              "Back to the row"ed)
          "Next row"5v+i))
#

@2"Set predecessors in the assigned rows to 2
                                     (from <)"
"  SETUP"
("Top of matrix"1^uu
 "Set class number model"to000$j11^&ut

"  PROCESS"
1^("WHILE rows remain"
        ("IF in the current position class"3pc?
         "THEN on row"4k
           1^("WHILE positions remain"
                   ("IF a one"pil$?
                    "THEN convert to 2"&)k))
         "Next row"i))
#
```

Intermodule connections

The following table of subroutine calls was prepared by Alan L. Teubner.

Name	Calls	Called by
C	i,f,o,m,s	none
A	none	I,M,r
B	b	i
b	none	B,i,J,m,o,r,s,V,<,>
D	p	f,I,j,K,M,m,N,o,P,s,Z,<,=,\,~
d	N	f,M,r,Z
E	N	f,Z
F	none	s
f	D,d,E,L,M,P,r,S	C
G	none	k
g	none	I,i
H	none	i
I	A,D,g,y	i
i	B,b,g,H,I,J,0	C
J	b,y	i
j	D,k,N	s
K	D	s
k	G	j,Z
L	n,p,z	f
M	A,D,d	f
m	b,D,P,R,V,Z	C
N	D,~	d,E,j
n	none	L
o	b,D,P,<,>	C
P	D	D,f,m,o,s
p	z	L
R	T	m
r	A,b,d	f
S	none	R
s	b,D,F,j,K,P,W	C
T	none	R
V	b,w	m
W	none	s
w	none	V
Y	none	Z
y	none	I,J
Z	D,d,E,k,S,Y	m
z	none	L,p
<	b,D,2,=,\	o
>	b,1,=,\	o
=	D	<,>,\
\	D,=	<,>
~	D	N
0	none	i
1	none	>
2	none	<

The conceptually significant subroutine structure is as follows. Other subroutine calls not listed here are for output or for general housekeeping.

```
C Main procedure
   i Initialize
      B Clear buffers
      H Open output
      I Find which morphemes to analyze
      J Find which processes to use
   f Read in forms
      r Read one data line
      L Look up a morpheme and build tables
         n Test if new
         p Add to list
      M Print the morpheme list
   o Positional analysis
      > Analyze predecessors
         1 Flag assigned columns
      < Analyze successors
         2 Flag assigned rows
   m Distinct sets
      V Fill the union matrix
         w Complement the union vector
      Z Distinct sets
         Y Hidden complete sets
      R Form a reduced matrix
         T Eliminate columns
   s Component subgraph specifications
      K Look for a zero vector
      W Count the weight of each column
      F Find the lowest column sum
      j Print specifications for that column
```