COMPLICATED LANGUAGE DATA IN COMPUTABLE FORM

Daniel G. Hays
University of Missouri--Columbia

This paper describes some features of a system for preparing and storing language data in a digital computer in such a form that it is readily available for examination and analysis. The system, called ACTS, for Arbitrarily Complicated Text System, allows the language analyst considerable flexibility both in the use of conventions and in being able to represent in the machine interrelated data of various sorts.

ACTS was designed to handle heavily annotated transcriptions of discourses, and related data, though it may be used of course with language data that have a simpler form. It might be used to store a set of newspaper articles, for instance, or even a set of unrelated sentences; though it is most useful when data of diverse types are interspersed in a corpus.

In its present state of development, ACTS consists of (1) routines for text input, including routines for handling symbol conventions constructed by the linguist, (2) programmed procedures for segmenting the text and inserting structural labels in its internal representation, and (3) procedures for setting up directories which allow the linguist to get to various parts of his data easily and without the tedium of human bookkeeping. It is programmed in PL/I and runs on an IBM 360/65 computing system, with heavy reliance on disc storage.

The Character Representation Problem

One of the annoying aspects of using the computer with language data is that with present machines, there are usually not enough symbols available, or else they are not just the ones which are needed. Standard computer input and output devices--card punching machines and high speed printers--have enough characters to satisfy a person adding some numbers, or a businessman keeping a parts inventory, but you are not likely to see a schwa represented on one, and there are many other characters which linguists are fond of which will not be found there either.

Except for some slow-speed devices, or else some extremely expensive photocomposition and picture drawing devices which may not be available, the language analyst who wishes to use the computer must make some compromise with his need to see the graphs he is used to looking at, and to accept some conventions for equivalent representations using the available sets of characters. Often, especially in the case of "exotic" languages, conventions for computer representation do not exist and must be invented by the language analyst himself.

ACTS has facilities for defining equivalences among input symbols, internal representations, and characters in the output media. These facilities provide more flexibility than is usually available in working with strings of characters.

For reference, Figure 1 lists the character sets available on some standard IBM input and output devices. In many computing centers, the IBM 029 cardpunch is the standard data preparation device, and a printer with 60-character print chain in the standard output device. Neither have lower case letters, among other characters which might be useful.

If it is possible, it is desirable to use a typewriter-like device for preparation of text for entry into the computer, such as a remote terminal linked by telephone lines to the computing system, or a device such as the IBM MTST, which records a character code on a magnetic tape cartridge every time the typist hits a key. Since "bouncing balls" for Selectric typewriters have a wide variety of character sets, including at least one version of phonetic script, the symbol problem for input is not so serious as it once was.

Output is a more serious problem. If a remote typewriter terminal is used for input, it can be used for output as well; but this convenience is expensive, and may be prohibitive for the output of large amounts of language data. For a language analysis project which involves the computer, it seems likely therefore that at least some of the time, line printers, with conventions for the representation of one linguistic symbol by a different available symbol--or some combination of the available symbols--may be desirable.

Because of the above considerations, ACTS has several character conversion facilities in the programs for taking in text and printing it out. Four classes of correspondence can be specified.

First, there are one-to-one mappings of input character code to internal storage character code, and to output character code. For example, suppose your bouncing ball has the @ character in the usual position, but you do not need it. You want a vertical bar character instead. So you specify a one-to-one character conversion from @ to the internal representation of the vertical bar. You will then get a vertical bar on the high-speed printer on output, automatically, if that character exists on the print chain. If you are using a 48-character chain, which does not have the vertical bar, you could specify a one-to-one mapping from the internal representation to the slash character.

Second, it is possible to specify a unary shift character which allows digraphs in the input which contain the shift character in the first position to be converted to single character representations internally. For instance, if * is defined as a unary shift, you can specify that *(is to be converted to the internal representation for a square open bracket. A reverse mapping, from a single character internally, to an output digraph, is also possible.

In addition, bracketing shifts may be used. Here, an upshift-

Input device character sets:

1. IBM 029 card punch.
   0 through 9 A through Z . , : ; ? - ( ) / ' " _
   * & @ # $ ¢ % + = < > not-sign, vertical bar

2. Typewriter-based devices.

   Varies, especially with Selectric typewriter devices.
   Ordinarily, . , : ; ? - ( ) / ' " _ * & @ # $ ¢ % + =
   plus 3 or 4 other special characters.

Output device character sets:

1. 48-character print chain on high-speed printer.
   0 through 9 A through Z
   . , - ( ) / ' * & $ + =

2. 60-character print chain.
   All the above, and : ; ? " _ @ # % < > not-sign,
   vertical bar

3. 120-character print chain.
   All the above, lower case letters, superscript
   numerals, superscript + - ( ), square brackets,
   one-line braces, dash, plus-minus sign, degree
   symbol, exclamation point, round bullet, square
   bullet, and a few others.

Figure 1. Sets of Characters Available on Representative Input and Output Devices

character and a downshift-character are defined. Characters within the pair are treated in a special way. Often, bracketing shifts are used to lead to a representation of some other alphabet, or may with punchcard input indicate underlining.

Finally, overprinting can be handled. If % is defined as an overprint indicator, the string 0%/ could be converted into a representation of the symbol Ø. On output, this representation could be specified to lead to an actual overprint by the line printer.

Structure of the Basic Text File

The symbol conversion routines described above may be used independently of the rest of ACTS, but they are only preliminary to the second phase of processing, during which the text, with its input peculiarities straightened out, is segmented and labelled. The result of this phase is a data representation in computer storage called the basic text file. In this phase, cues which are present explicitly in the text, such as symbols which the user defines as brackets for a type of data, as well as surface contextual cues, are used by the program. The program does not use information from the syntactic or semantic structure of the natural language which is represented, but relies only on the "graphic syntax" and related interpretations supplied by the user for his symbol strings.

Let us take a simple example of what is meant by segmenting and labelling a text. In the sentence

It's a cold day in October, honey-pie.

the basic units may be taken to be graphic words, non-terminal punctuation marks, and terminal punctuation marks. The graphic words are bounded by blanks or by punctuation. The sentence itself is a higher order unit bounded on the left by the boundary of the whole text, and on the right by a terminating punctuation. This example, though simple, is not without some problems. The apostrophe in "It's" must be distinguished from possible occurrences of the same symbol in quotation pairs, and there is an option of whether to treat "honey-pie" as one word or two words, or perhaps a special data type for compound word.

Any time the computer is used to process the sentence in its bare, unsegmented form, let us say for purposes of dictionary lookup, it must provide tests to do enough segmenting to, for instance, separate words from punctuation and determine that some apparent punctuation characters are actually part of the word. Human beings do this sort of task easily and naturally, but computers are notoriously "literal minded."

If the sentence occurs along with a number of others, and one is interested in picking out one sentence at a time, the machine must scan through the text looking for terminal punctuators. This is simple enough to do, but does require comparing every character in the text with a list of legal terminal punctuation characters. If the punctuation "..." occurs, the program must perform a further test every time the character "." is encountered.

However, if the text is segmented once and marked, it does not have to be resegmented each time it is examined. Computers are fast, but they are expensive, and in the long run unnecessary operations mean unnecessary expenses. Furthermore, the kind of text segmentation and marking that is performed by ACTS leads readily to the construction of directories stored separately from the text itself, which allow rapid and efficient entry into the text by data-type, and even by content for a given type of data. It is as if one had a text and the functional equivalent of a concordance stored in the computer along with it.

Not all text which a language analyst might be interested in consists only of sentences strung out one after the other, however. Especially if the investigator is interested in descriptions of language and language-related behavior as it occurs in real situations, more kinds of data will be reflected in the transcript--and will need to be sorted out. Such sorting is very tedious for human beings. Utterances have speakers. Utterances are often directed to other speakers, possibly in the presence of other people. People make gestures, and do other things when they are speaking and in between times they are speaking. The language as it comes out of the mouth is subject to what are often ignored in linguistic investigations as performance peccadilloes--yet these actually occurring forms may be important to note before a too-hasty translation and reduction to the investigator's version of Standard Form.

Other kinds of data may be interesting as well. The class of situations or physical settings in which a discourse occurs, and in some cases the genre of the discourse--which may be particularly relevant for written language but also I think has some applicability to spoken language--these may also be important to note. Information about objects or events in the situation which are referenced by a language user may be important also in analyzing the text. There are other kinds of data which language analysts have examined at one time or another, such as prosodic data of various sorts. Also, it is often helpful to include incidental and unsystematic observations or annotations along with the basic data, to indicate insights, peculiar circumstances, or tentative interpretations. Months after the data were gathered, a note to oneself will help recall just what was going on, whereas the bare data might only be puzzling.

In the current version of the data-structuring program in ACTS, the investigator may define as many as 255 data-types, and specify cues and contexts for their recognition. Included in this number, which is arbitrary but convenient for technical reasons when working with IBM computers, are both <u>elementary data-types</u> and <u>higher-order data-types</u>.

What is elementary and what is not depends on the choice of the investigator, and his formal specifications for his data structure. One man's elementary type may be another man's higher-order type. In the example of language-switching text in Figure 2, "a word of a language" is taken to be an elementary data-type, whereas in the example of phonetic text in Figure 3, "word" is a higher-order data-type subsuming word segments and some other kinds of data such as stress indicators.

Before Character Conversion

```
:Juanita:  *S Pero *E what...what I think about that is they're way
                out there.
:Maria:  *S Pero lo gastas todo al gas.  (under Juanita)
:Juanita:  *E I think, *S yo gasto ma/'s gas...ma/'s gas en ir pa'
                'lla/' a comprar esa cosa barata cuando podi/'a
                comprarla aqui/'.
```

Third Actor Block
After Character Conversion

```
:Juanita:  *E I think, *S yo gasto más gas...más gas en ir pa' 'llá a
                comprar esa cosa barata cuando podía comprarla aquí.
```

Figure 2. Language Switching Text Example

(He) ^{1}wǽt^{4}↑ ^{2}yəspɛ̀nt ^{3}fâiv ^{4}hə́ndrə́d^{3} ^{2}dalə̆rz^{4}↑

(She) ^{4}néw^{2}↓ ^{2}fáiv^{3} ^{4}hə́ŋriy^{3} dálərz^{1}↓ ^{2}stûw^{3}pɨd^{3}→

Figure 3. A Phonetic Representation

The kinds of cues which are used in the recognition of data-types are of several kinds. First, the internal character set is sorted into several classes, such as alphabetic characters, numeric characters, and special characters whose function may be recognized by their class membership. Second, symbols or strings of symbols are specified as explicit brackets for certain data-types. Third, left and right contexts, above and beyond explicit bracketing symbols, are defined in terms of the basic units and higher order units which will have been recognized in earlier stages of processing.

In internal storage in the computer, each token of an elementary data-type is marked with a left-boundary marker so that it can be found when examining the text sequentially, a numerical code for the type, and a tag which contains information of use to the system, such as information about graphic connectibility and perhaps other information supplied by the user. Higher-order data-types, such as sentences, actor block, etc., are marked in a similar fashion, and their contents are the concatenation of all lower-order data tokens in their range.

Thus, the basic text file might be thought of as an "invisible" labelled bracketing of the raw text, where the bracketing is determined from a graphic syntax. This characterization is complicated by the possibility of discontinuous units, however. The data structures are not restricted to trees, since, for instance, sentences can start in one speaker block, and end in another. For this reason, the recognition routine is a bit involved, but works as it were from the top down when explicit bracketing strings are given, and from smaller units to higher-order ones when more complicated contexts are involved.

Let us consider some possible structuring of linguistic texts. For the example of code-switching given in Figure 2 (which was kindly supplied by Professor Donald Lance), the structuring program would, in addition to segmenting words and punctuation marks, put "invisible brackets" around each stretch of text belonging to one speaker, mark the beginnings of sentences, and mark the annotation "(under Juanita)" in a special way so that it can be retrieved easily along with other annotations which begin with the keyword "under". In addition, passages marked with the *E or *S left context would be marked according to the language, and each text word in the passage would be marked also for its language.

For the example of a phonetic representation of a brief dialogue, given in Figure 3 (which was supplied by Harriett Nutt Hays), a somewhat more complicated structure is called for.

It is assumed for this example that the necessary special characters exist on your typewriter-like input device, and that suitable character mappings have been established between the input code and the internal code. Stress symbols are in the input preceded by an "invisible" code for a backspace (which is treated by the machine as a character). This backspace code is used by the data structuring program to identify the occurrence of a segment belonging to the "stress mark" data-type. The numerals in the input are recognized as belonging to the data-type "pitch symbol" simply by virtue of their being numerals. The terminal contour data-type is similarly recognized.

The elementary data-types for this example might be:

a. phonetic segment, consisting of strings containing adjacent characters representing vowels, consonants, semivowels, etc.
b. stress mark
c. pitch symbol
d. terminal contour indicator
e. open speaker designation bracket
f. speaker designation string
g. close speaker designation bracket.

The higher-order data-types would be:

h. phonetic "word", consisting of phonetic segments, stress marks, and pitch symbols bounded by blanks or possibly by a blank and a terminal contour indicator
i. phonological "clause", consisting of strings of phonetic "words" bounded on the right by terminal contours
j. speaker designation units, consisting of data-types, e, f, and g in sequence
k. speaker blocks, consisting of everything from the beginning of a speaker designation unit to just before the start of the next such unit (or end of the text).

With the text structured and marked internally in this way, data-accessing directories can be constructed that allow access to the data at various levels. Accessing the data by the content of various types is also possible.

Finding the Data

A more technical discussion of the mechanisms used in getting to the data in a somewhat earlier version of ACTS is given in Hays (in press). Technical details are not important for this presentation, but the fact that there are two basic ways of accessing the data may be helpful in understanding the system.

In one kind of access file, the locations in the basic text file of tokens of a given data-type, or set of data-types, are listed one after another. This kind of directory is of most immediate use when the researcher wants to quickly pick out higher-order units such as sentences, or to locate tokens of types which occur infrequently and are hard to see when scanning the text by eye. In the second kind of access file, the contents of the occurrences of the data-type, in the case of elementary units, or the contents of some elementary unit within a given set of higher-order units are sorted, allowing the text to be entered by content. If the user is interested in the occurrences of English "function words", for instance, he could give the machine a list of function words and find the sentences in which they occur. Or, all utterances spoken by a specified actor could be located directly.

In constructing the access files, the user can choose which datatypes are to be grouped together for retrieval purposes, and he can also choose the kinds of data he wants to reference by one means or the other.

Summary

Some features of ACTS, a computer-based system for handling language data, have been described. This system has facilities for symbol conversions, to aid the linguist in setting up conventions for representing his data within the limitations of sets of characters on computer input and output devices. In addition, it allows rather elaborate structuring and labelling of the basic data, so that the data may be more easily retrievable.

REFERENCES

Hayes, Daniel G. In press. 'Accessing information in behavioral description analysis.' In *Proceedings of the Seventh Annual National Information Retrieval Colloquium*. Philadelphia, College of Surgeons.