Exploration Schemes in the Linguistic DataBase

Drs. Hans van Halteren, University of Nijmegen

Introduction.

The *Linguistic DataBase* (*LDB*) is a database program for the storage of tree structures [6],[7]. Its main use is the interactive exploration of syntactically analysed corpus texts for reasons of linguistic research. It was developed, primarily for the last purpose, by the TOSCA group at the University of Nijmegen. This group is currently producing analysed corpora of English, Spanish and Arabic [1],[3],[5],[11]. A further project, in which Hebrew will be studied, starts in the second half of 1989.

The *Linguistic DataBase* itself is not able to produce a syntactic analysis of text. For this it relies on other programs. At Nijmegen we feel that in the near future no analysis program will be able to analyse text correctly without any human intervention. The corpus texts we work with have shown us that no grammar that is practically usable will ever be complete enough to deal with the variety produced by native speakers. Additional problems are caused by the ever-present ambiguity of natural language [12]. This means that throughout the analysis of a corpus text, we have a linguist present to supervise, to help out in exceptional cases and to disambiguate where necessary [8]. Once the material has been analysed it can be explored with the *LDB*.

The syntactic analysis trees stored in the *LDB* represent the constituent structures of the utterances in the corpus. Each node stands for a constituent and contains labels for the syntactic function, the syntactic category and for any other attributes of the constituent. For example, in the utterance '`A man walked by.`' the node representing the constituent '`A man`' would have the function label 'subject', the category label 'noun phrase' and attribute labels 'third person' and 'singular'.

In the *LDB* these trees can be examined on a normal terminal screen with the resident *tree viewer*. As will be shown in the examples below, the tree viewer shows the trees as 'growing' from left to right. This is done to make more efficient use of the space on an all-to-small screen. The user can move a so-called *focus* around the tree to get information on the specific nodes that are of interest to him.

For exploration across more than one tree the *Linguistic DataBase* provides *exploration schemes*. These can be used to search for analysis trees with specified configurations of nodes, but also to extract information, eg. the texts of all constituents of a certain type or a frequency count of some syntactic phenomenon. Exploration schemes consist of two parts:

- a pattern describing what is to be looked for;
- instructions for what has to be done once it has been found.

As the exploration schemes can become quite complex, the *LDB* provides a specialized *scheme editor*. This editor uses the same method of presentation and the same commands as the tree viewer for maximum consistency and consequently ease of use.

The best way of showing the possibilities of a computer program is by means of a number of examples. As it is impossible to show the interactive aspects of the *LDB* in a paper presentation, the examples will focus on exploration schemes. The variety of specifiable properties will be demonstrated by a series of exploration schemes concerned with constituents functioning as subject complements. As the purpose here is to demonstrate what can be done with the *LDB* the linguistic importance of the results of these schemes is negligible.

Each example is presented by two or more figures. These figures are screen dumps of the *Linguistic DataBase* during an interactive demonstration. They show the situation as would a VT100-compatible terminal screen. The example is then discussed in three paragraphs. First the the features new to the exploration scheme are described. A description of what the scheme as a whole is supposed to do follows. Finally, the result of executing the scheme is explained.

The above-mentioned analysis projects are not yet in a stage where large amounts of analysed utterances are ready for publication. Therefore, the examples have to be based on the results of the early Computer Corpus Pilot Project (CCPP). In this project a rather small corpus (130.000 words) was analysed syntactically, with the use of a context free grammar [9]. The 'prehistoric' nature of the project shows itself in the presentation of the utterances: upper case only. The use of a context free grammar means that no attribute

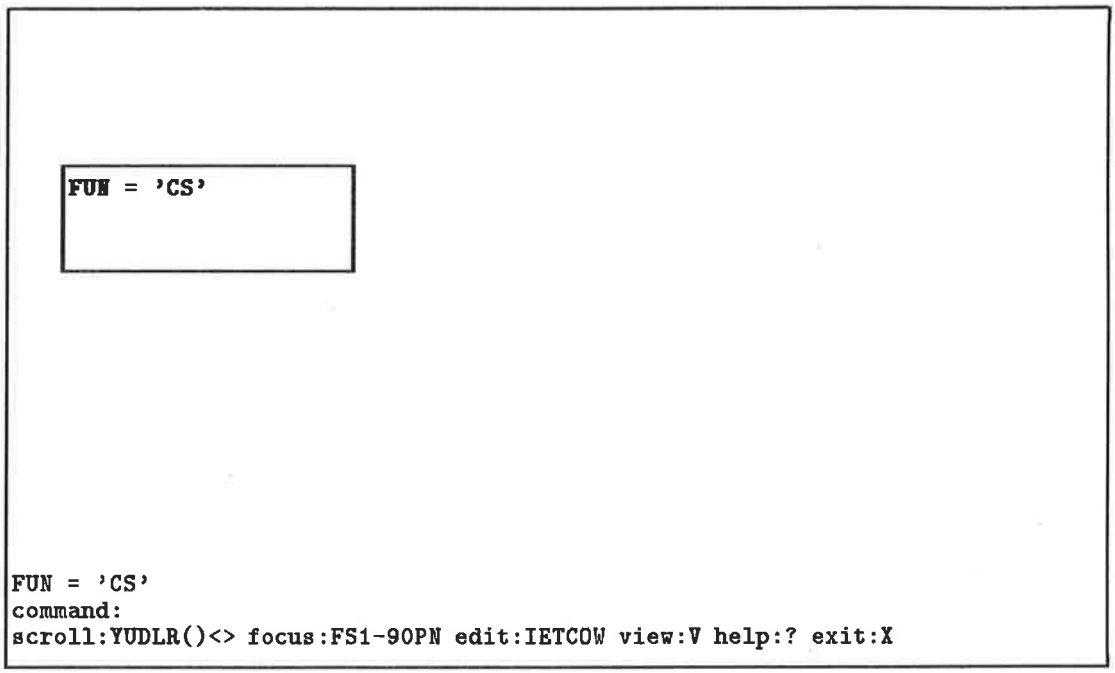

Figure 1a.

labels are present in the data. These two facts are the reason that intelligent character-level matching and attribute handling can not be demonstrated.

Example 1: searching for nodes.

The first example concerns a very simple exploration scheme which finds and shows all subject complements. Figure 1a shows the search pattern of the scheme in the pattern editor.

Somewhere in the middle of the screen is a box. This is the *search pattern*, a tree structure to be matched with part of an analysis tree. This particular pattern consists of only one node, which means that only one node of an analysis tree will be necessary for a match.

To restrict the analysis nodes matched by a pattern node it is possible to put one or more *specifications* on it. Here we find one specification: `FUN = 'CS'`. The specification must be correct for an analysis node, i.e. the expression forming the specification must yield the value `TRUE` when using the properties of that node, in order for it to match.

`FUN` is a *specifier* and stands for the function label of the analysis node under consideration. The function label is a string.

In `'CS'` the single quotes are delimiters indicating a string *constant*. This string, `CS`, is an abbreviation of subject complement. The reason that `CS` is surrounded by single quotes and `FUN` is not is that `CS` is not a keyword of the Linguistic Database program. The abbreviation `CS`, the name subject complement and even the fact that subject complement occurs as a function label are properties of the data.

`FUN` and `'CS'` are connected by = which is a *comparison operator*. The comparison operator = demands equality between its operands, in this case strings.

Apart from the tree structure forming the search pattern there are three more lines of information on the screen. The first line shows the contents of the the focused node. As there is only one node present this is the specification `FUN = 'CS'`. The second line is the *command line*. It is reserved for the prompt `command:`, the echo of the typed commands when in echo mode and error messages when commands are invalid. The bottom line shows an (abbreviated) list of all available commands.

The *match effect* of this exploration scheme, which is not shown in a figure consists of the single *action* `USER`. This means that whenever a match is found the program will give control to the user for examination of the analysis tree found.

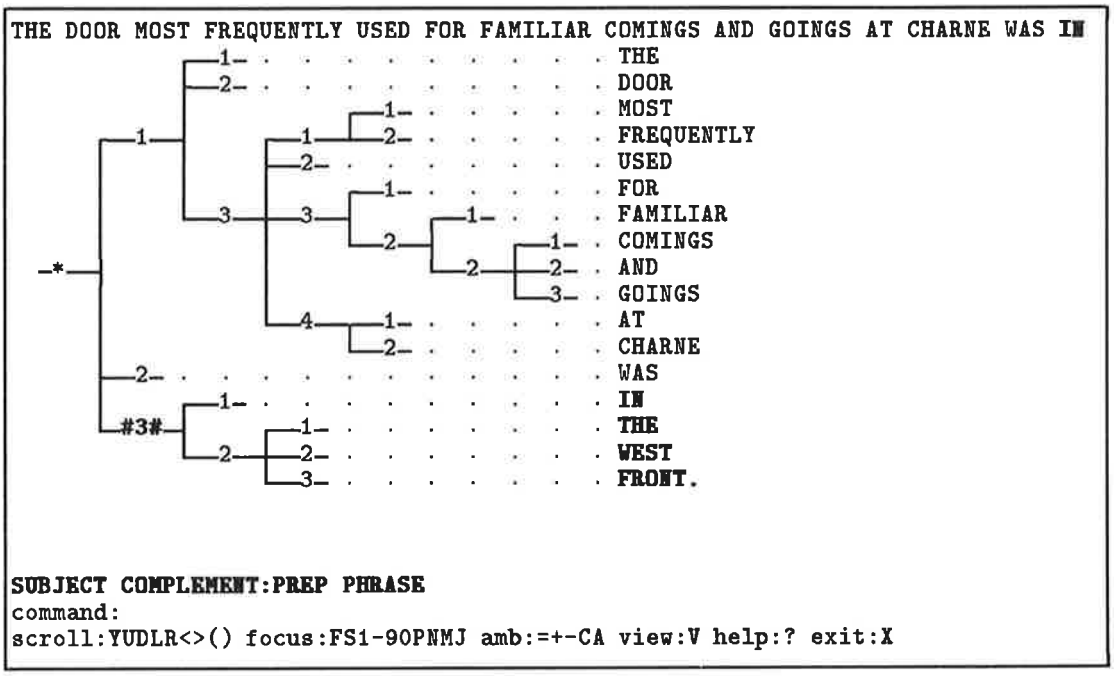

Figure 1b.

The exploration scheme looks for all analysis trees containing a node with the function label subject complement. Whenever it finds one, it marks the matching node and stops. The user can then use the tree viewer to inspect the matching tree.

The exploration scheme stops and, after the focus has been moved to the matching node, the tree viewer presents the screen as shown in figure 1b.

The tree viewer is in the *tree-map view*. In this mode the tree structure is shown in full, but the node labeling is absent, apart from the labeling of the focus, which is shown on the third line from the bottom. The last two lines of the screen are again the command line and the list of available commands. The top line shows the words of the utterance.

The numbers representing the nodes in the tree have meaning only for focus movements (eg. the 3-key was pressed to move from the root of the tree to the current focus). The matching node is marked with a two hashes '#'. The focused node is highlighted in the tree, as are the corresponding words to the right of the tree and at the top of the screen.

The subject complement is this utterance is IN THE WEST FRONT.

When the v-key (the change-view command) is pressed the tree viewer switches to *environment view*, shown in figure 1c.

The tree viewer has zoomed in on the focus. All but the direct environment of the focus has been lost, but now the (abbreviated) labeling of all the shown nodes is also visible. In each node there is a function and a category label. The empty line is reserved for extra information, the so-called *attributes*. An example of an attribute would be singular for the noun phrase THE WEST FRONT. However, the CCPP analyses do not use attribute labels. The focus is still indicated by highlighting. The matching nodes can be recognized by an M in the top left-hand corner.

Only the tree area of the screen changes with the viewing mode. The other information lines, and all commands, remain unchanged.

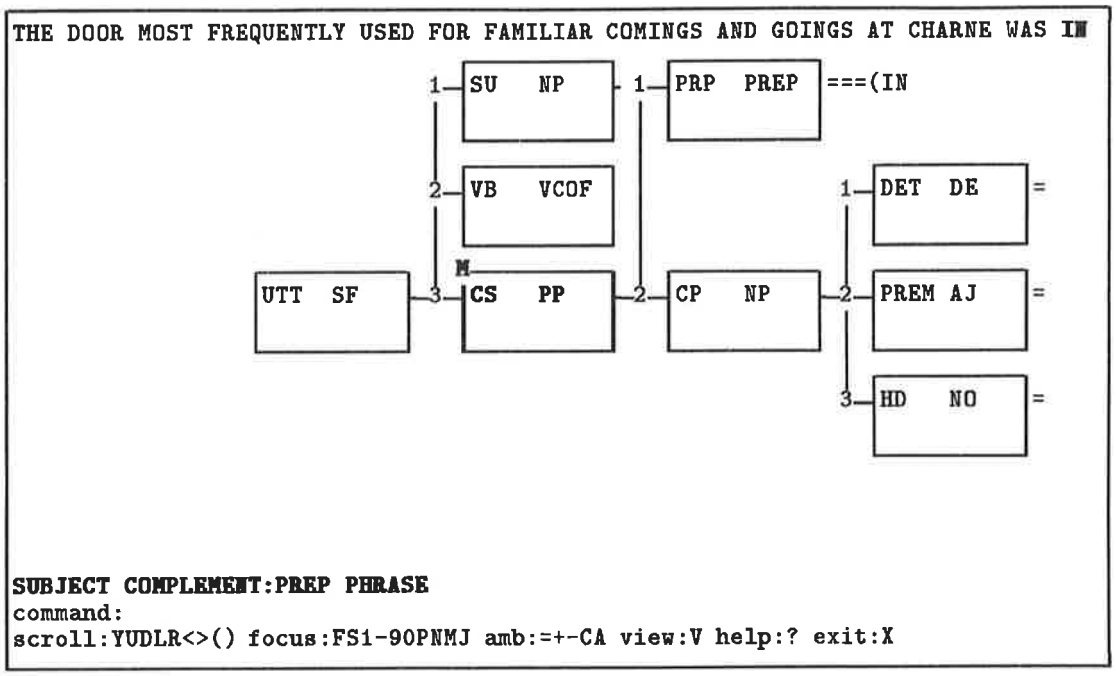

Figure 1c.

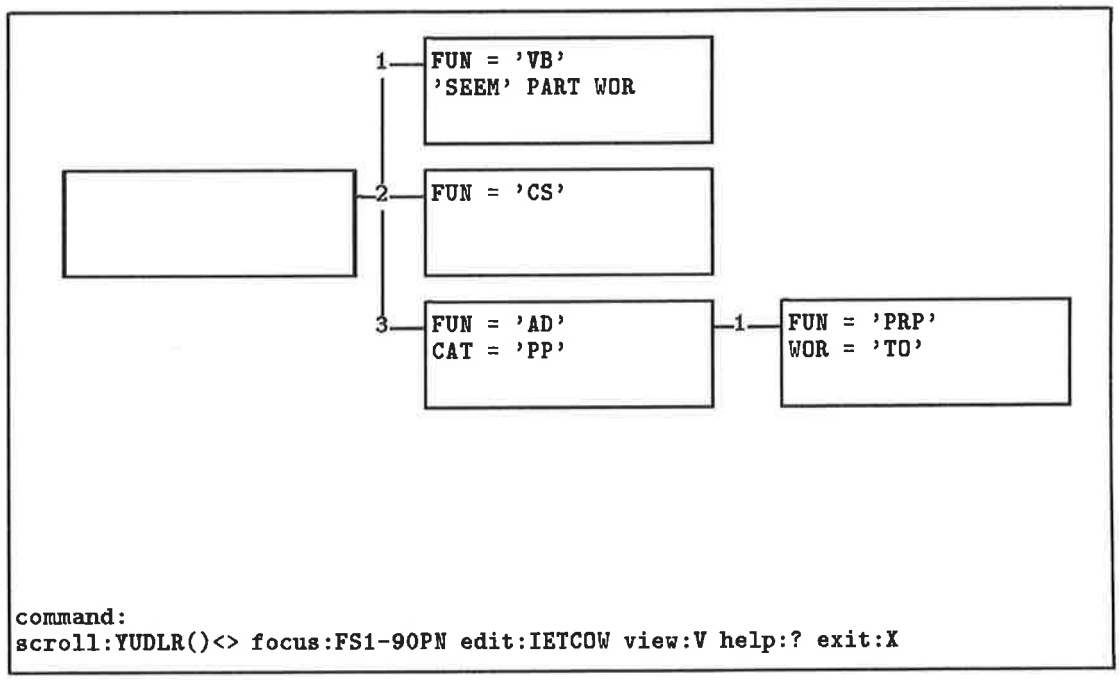

Figure 2a.

Example 2: searching for configurations.

The search pattern becomes more complicated in order to find only a certain type of subject complements. The new pattern is shown in figure 2a.

The search pattern now consists of more than one node. There are three rules that govern the possibilities of matching such a pattern:

- immediate descendants in the pattern tree correspond to immediate descendants in the analysis tree
- the order of the descendants of a node is significant

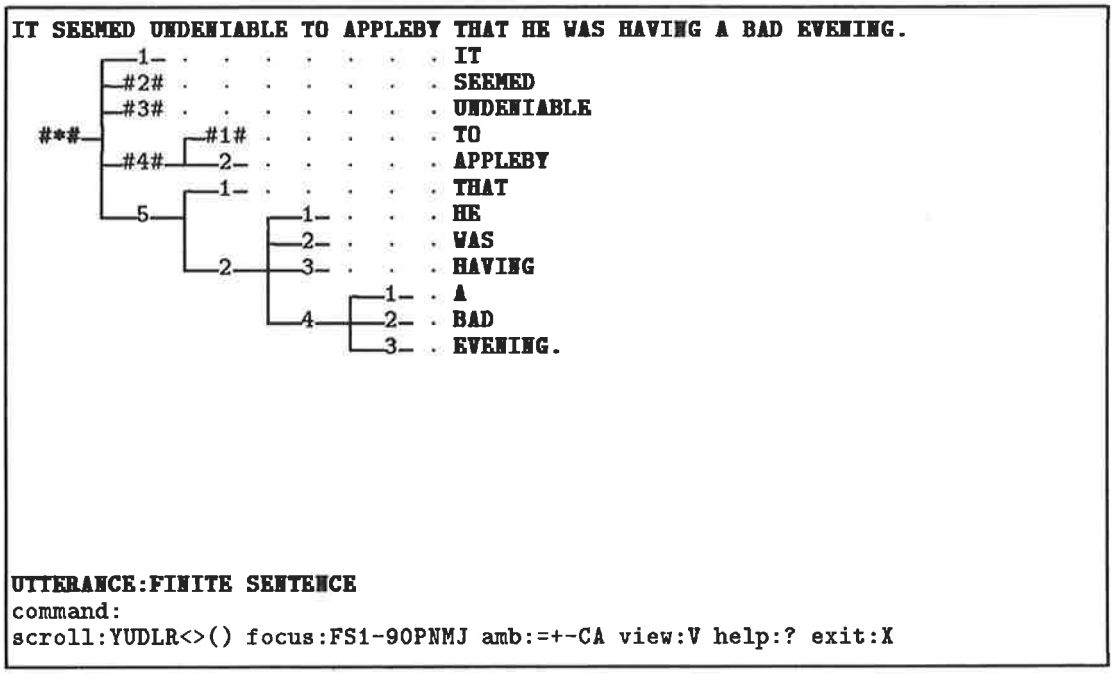

Figure 2b.

• adjacency of two descendants in the pattern does *not* necessarily entail adjacency in the analysis tree

Note that the last 'rule' implies that not all descendants need to be present in the pattern. How each of these rules can be adjusted will be shown in some examples below.

Some pattern nodes now contain more than one specification. To get a match all specifications must be correct.

The specifier `CAT` is similar to `FUN`. It stands for the category label of the node.

`WOR` is yet another string specifier, representing the entire text belonging to the node, i.e. all the words of the constituent, separated by blanks. In case the node concerned is a leaf of the tree, this is just the one word at the leaf.

There is also a new comparison operator, `PART`. It yields `TRUE` when its left (string) operand is a substring of its right (also string) operand. Here the specification says that the string `SEEM` must be part of the words of the constituent, eg. `SEEMS`, `UNSEEMLY`, `WOULD SEEM NOT TO GO` or, of course, `SEEM` itself.

The new abbreviations in the labels are `VB` for verb, `AD` for adverbial, `PP` for prepositional phrase and `PRP` for preposition. `SEEM` and `TO` are not abbreviations, but just words or parts of words. Note that the use of upper case is a feature not of the software, but of the data.

As in Example 1 the exploration scheme looks for an analysis tree containing a subject complement, marks it and stops so that the user can use the tree viewer to inspect it. However, this time there has to be a form of 'to seem' functioning as a verb on the same syntactic level as the subject complement and preceding it. Furthermore, still on the same level and following the subject complement there has to be a prepositional phrase with the preposition 'to', functioning as an adverbial.

When the first find of the exploration scheme is viewed the result is figure 2b.

The focus is still at the root of the tree, which is also the root of the matching subtree. All the nodes of this subtree are marked with hashes.

The verb is `SEEMED`, the subject complement is the adjective `UNDENIABLE` and the adverbial prepositional phrase is `TO APPLEBY`. In this tree the three descendants of the root happen to be adjacent, which is not actually necessary for a match. However, they are both preceded and followed by further descendants.

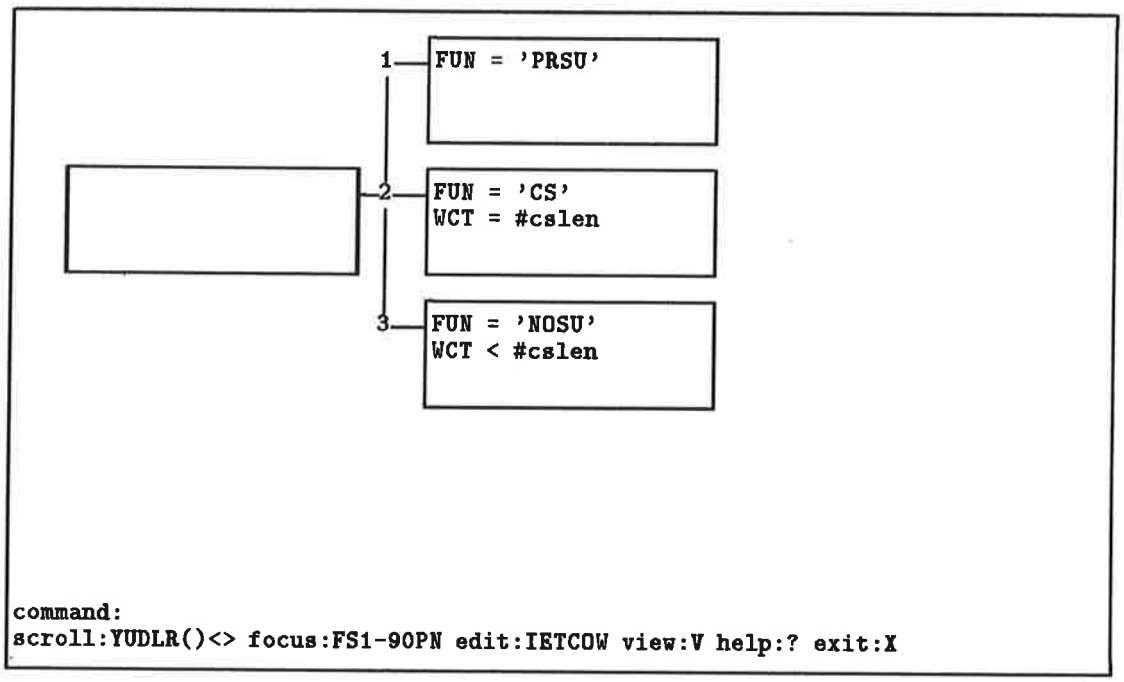

Figure 3a.

Example 3: pattern variables.

Again the class of subject complements that has to be found will be restricted by demanding accompanying constituents.

The main new feature of the pattern shown in figure 3a is the use of a *variable*. It is the numeric variable `#cslen`, its type indicated by the hash ('#'). Variables are something between constants and specifiers. Constants never change and specifiers change with the node they refer to. Variables change their value with the circumstances, but with regards to one (attempted) match they have the same value in all the places they occur in. So `#cslen` stands for the same numeric value in both the nodes where it is used.

The specifier `WCT` (word count) stands for the number of words of the constituent. It is not a string like the previously mentioned specifiers, but a number.

The comparison operator = is used here with numeric operands instead of strings, but it still demands equality. The new operator < indicates that its left operand should be smaller than its right operand. In case of strings this would mean precedence in alphabetical order.

The constants `PRSU` and `NOSU` are abbreviations for the function labels provisional subject and notional subject.

The subject complements found by this scheme are preceded on their syntactic level by a provisional subject and followed by a notional subject. Furthermore, the notional subject, measured in number of words, is shorter than the subject complement.

The exploration scheme finds the utterance shown in figure 3b. Again the matching nodes are recognizable by the hash-marks. The provisional subject is `IT` and the subject complement `IN THE NATURE OF PLANS` is indeed longer, with its five words, than the three-word notional subject `TO GO WRONG`.

Note that there seems to be a quote missing at the end of the sentence. This is, however, not the case as the rest of Bobby Angrave's speech will be found in the next utterance.

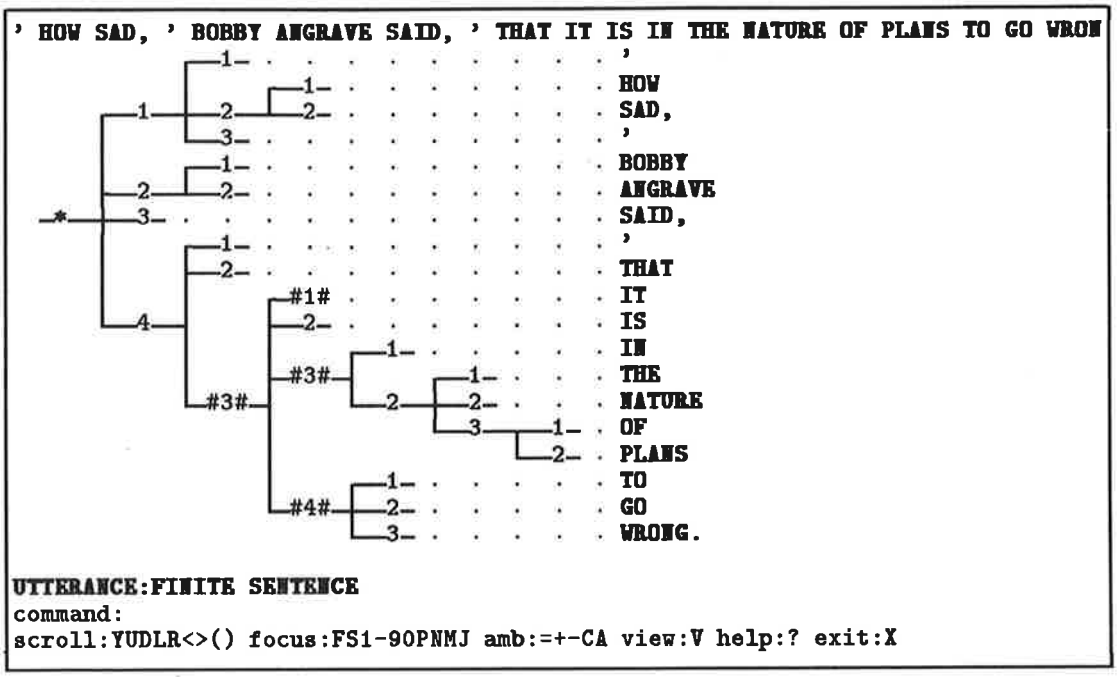

Figure 3b.

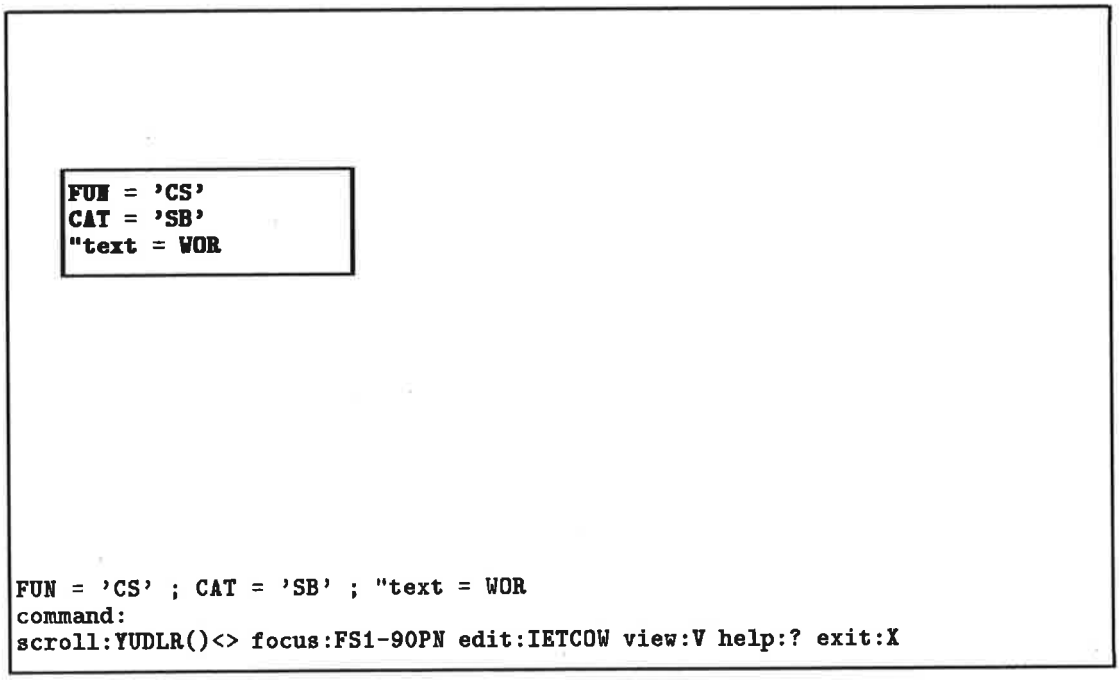

Figure 4a.

Example 4: the match effect.

In the fourth example the exploration scheme does not merely search for a certain type of subject complement. It performs an extra action, namely extracting data from the matching utterance, before handing over control to the user.

The pattern in figure 4a actually has no major new features. What is new is the use of a new abbreviation, SB, which stands for subordinated clause. Then there is the comparison between WOR and something called "text, which does not seem familiar. But "text is just another type of variable, namely a string variable.

```
EXAMINING CORPUS The Bloody Wood DERIVED FROM ccppbw

SCHEME: CS:SB, GOING FORWARD AT TREE 1336
TREE 1336 OF 1872   LOC: 0207128   WHAT DID APPEAR WAS THAT HE HAD ABRUPTLY LOS



 A select next tree                        M start a restricted corpus
 B select previous tree                    - insert current tree into new corpus
 C select tree by number                   - finish the restricted corpus
 D select tree by location                 P start exploration towards end
 E examine selected tree on screen         Q start exploration towards begin
 F display selected utterance              R continue exploration
 G display analysis comment                S stop exploration
 H print tree (all ambiguities)            T show codes in utterances
 I print tree (current ambiguity)          - don't show codes in utterances
 J print tree (current match)              Z go back to main menu
 K examine original corpus                choice:
 L examine label names



THAT HE HAD ABRUPTLY LOST INTEREST IN MRS GILLINGHAM.
```

Figure 4b.

However, there is something peculiar about the use of the variable. It occurs only once in the pattern and as such has no restrictive value. The reason is that the value of variables is accessible in the match effect, so that they can be used to ship information out of the pattern.

The match effect in this exploration scheme is:

```
SCREEN( NEWLINE, "text )
USER
```

After finding a match, these actions are executed in the stated order. The first action, **SCREEN**, sends information to the screen. Its parameters tell what has to be send: **NEWLINE** clears the available space and "text is the string provided by the pattern. After the information has been put on the screen the second action is performed which is the already familiar **USER** giving control to the user.

The exploration scheme looks for constituents with the function label subject complement and the category label subordinated clause. Every time it finds one, it takes the text of this constituent, puts it on the terminal screen and stops. The user, already seeing the text of the matching constituent, can now decide whether he also wants to see the tree structure.

A variation of this scheme would be to use **FILE** instead of **SCREEN** and not include **USER**. This would go through the entire corpus without stopping. However, it would put the texts of all matching constituents in an output file for further processing.

When the exploration scheme stops, the terminal screen appears as in figure 4b. This is the *corpus examination menu*. All actions outside the specialized tools like tree viewer and scheme editor are controlled through menus.

¿From top to bottom the screen shows an identification of the current corpus, an identification of the current exploration scheme, an identification of the current tree within the corpus, a list of all commands and the messages by the exploration scheme.

The commands are marked with letters or dashes ('–'). A letter means that typing that letter will execute the command. A dash means that this particular command is not available at the moment.

Typing the letter E results in the screen of figure 4c, which shows the tree in which the match was found. The structure of the subject complement **THAT HE HAD ABRUPTLY LOST INTEREST IN MRS GILLINGHAM.** can now be examined. However, if the internal structure would not be of interest, the previous screen already gave all the desired information.

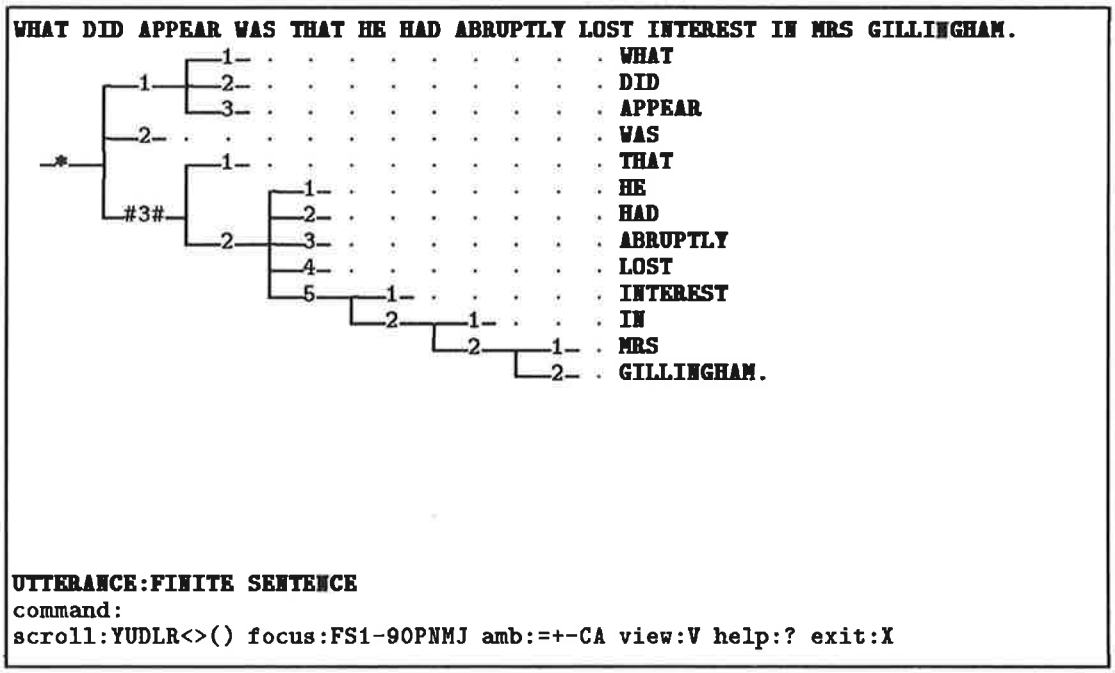

Figure 4c.

Example 5: further activities.

Possibilities of the actions accompanying the pattern but not directly responding to a match are introduced, as are conditional actions.

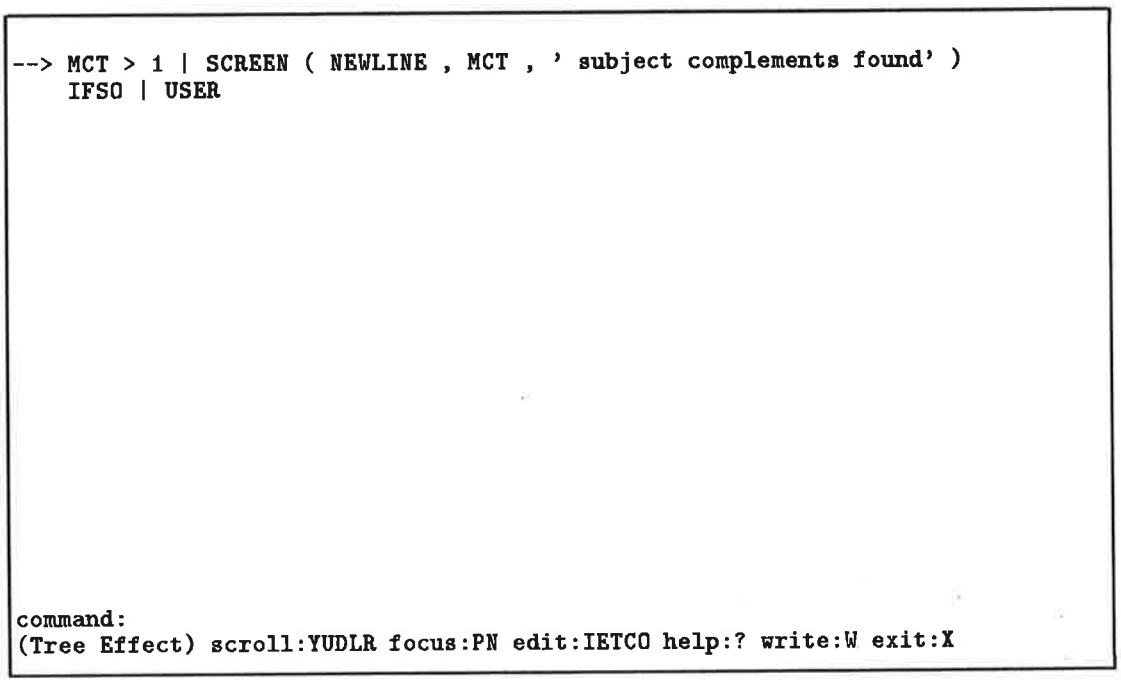

Figure 5a.

The search pattern of this scheme is the same as the one in example 1, consisting of one node only, with function label CS. There is no match effect.

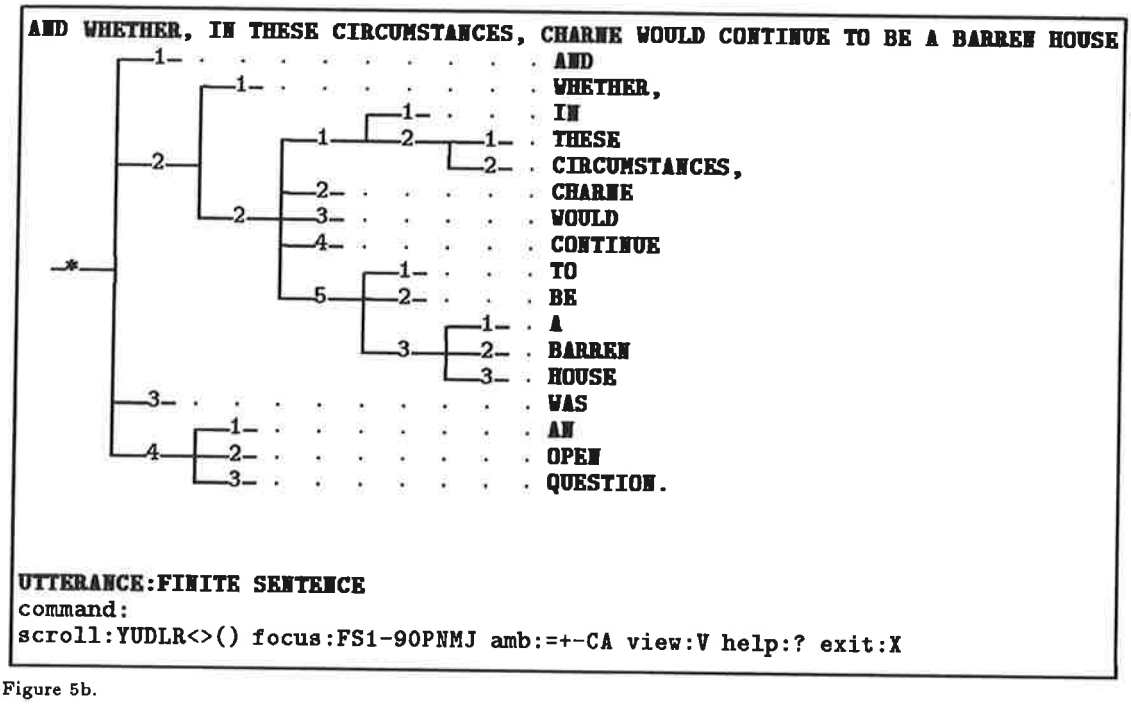

Figure 5b.

Figure 5a does not show the search pattern as the first figure of all previous examples did, but it shows the *tree effect*. The tree effect, just like the match effect, is an *activity*. Each activity has specific moments at which it is executed during a run of the exploration scheme:

- the corpus intro at the start of the run
- the tree intro when starting with a new tree
- the match effect whenever a match is found
- the tree effect when finishing a tree
- the corpus intro at the end of the run

So the tree effect shown here is executed every time a tree has been searched for matches.

The vertical bar ('|') in both the actions in the tree effect means that the action to its right will only be performed if the condition to its left is present. The special condition `IFSO` refers to the last condition that was evaluated. This can be used as an abbreviation mechanism in case of complicated conditions.

The specifier `MCT` (match count) acts as a kind of built-in variable. It stands for the number of matches already found in the current tree and can be used only in the match effect and the tree effect.

All this means that the first action checks whether there was more than one match in the current tree. If this is the case it produces screen output. The message line is cleared and the number of matches is given, followed by the literal string `subject complements found`.

The exploration scheme tries to find an utterance containing more than one subject complement. When it finds one, it reports the number of subject complements in the utterance and stops.

The exploration scheme is activated and when it stops, the tree is examined with the tree viewer (see figure 5b). There are no nodes which are marked with hashes, as there is no current match. All matches took place in the past, during the matching process for this tree.

However, some examination (facilitated by the fact that the tree viewer provides a command to move the focus to nodes with specific labels) identifies the subject complements. They are `TO BE A BARREN HOUSE` and `AN OPEN QUESTION`. The only relation between these two is that they appear in the same utterance, which is exactly what was looked for. An exploration scheme where a relation will be necessary is examined in the next example.

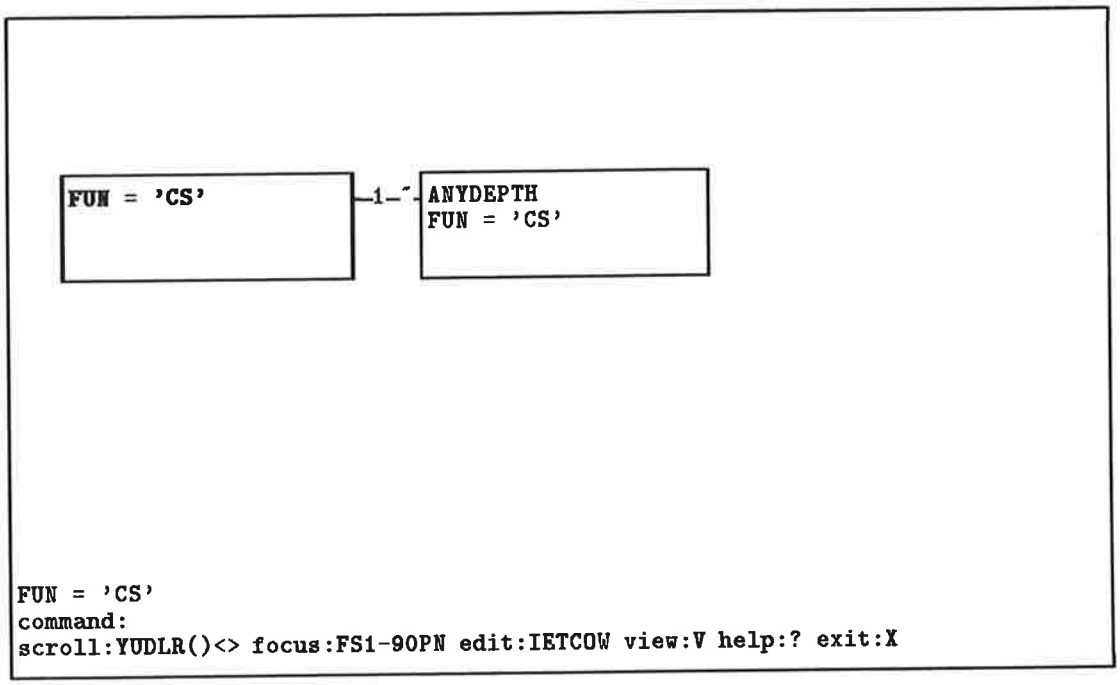

Figure 6a.

Example 6: overruling the defaults.

In this example, the 'rules' for matching parts of analysis trees are circumvented for the first time.

This pattern (figure 6a) shows only one novelty, but a very important one. As was mentioned above, the normal situation is that a connection in the pattern signifies immediate dominance. This restriction can be lifted by marking the dependant node with **ANYDEPTH**. **ANYDEPTH** is placed on the node just like a specification, but is called a *declaration*. It declares that the length of the connection in the analysis tree may be greater than one. However, there still has to be a dominance relation. To show the special connection in the scheme editor, it is marked with a tilde ('~').

The match effect has been reinstated, with the familiar single action **USER**.

This exploration scheme looks for utterances in which one subject complement is nested inside another. When it finds such a configuration, it hands over control to the user.

The tree found by the exploration scheme can be seen in figure 6b. The matching nodes are marked again, because the scheme stopped during the match effect. Of the two subject complements in this utterance, one (**CATCHING**) is indeed nested within the other (**THAT THEY SEEM TO BE CATCHING**).

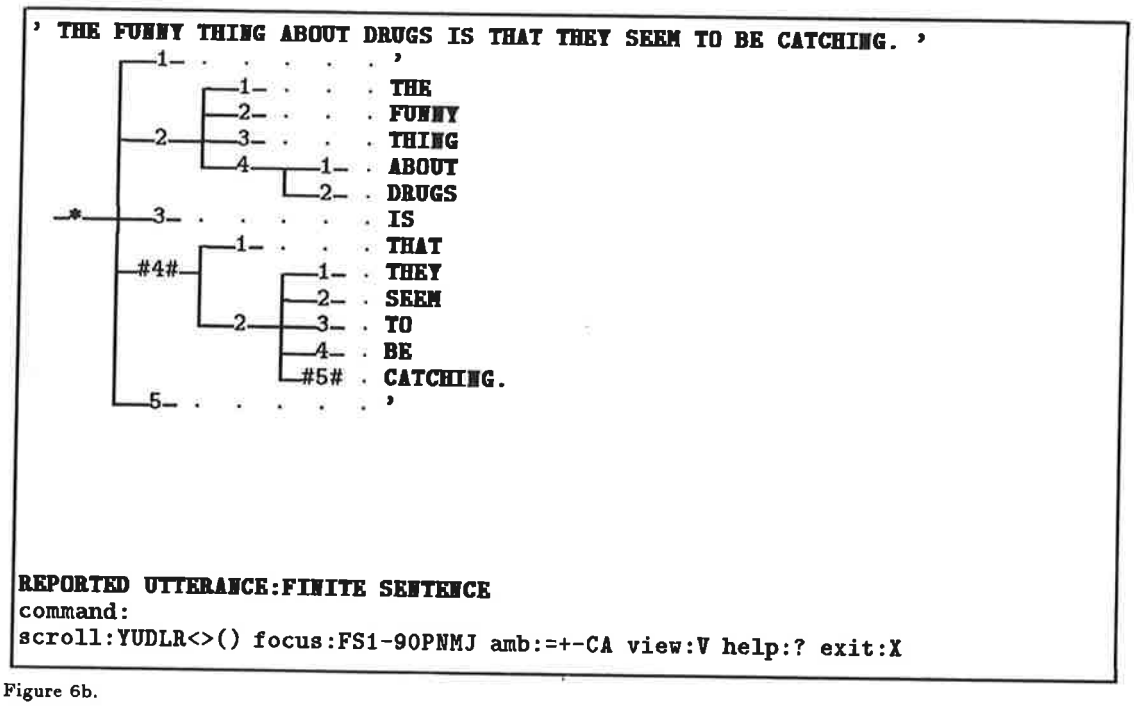

Figure 6b.

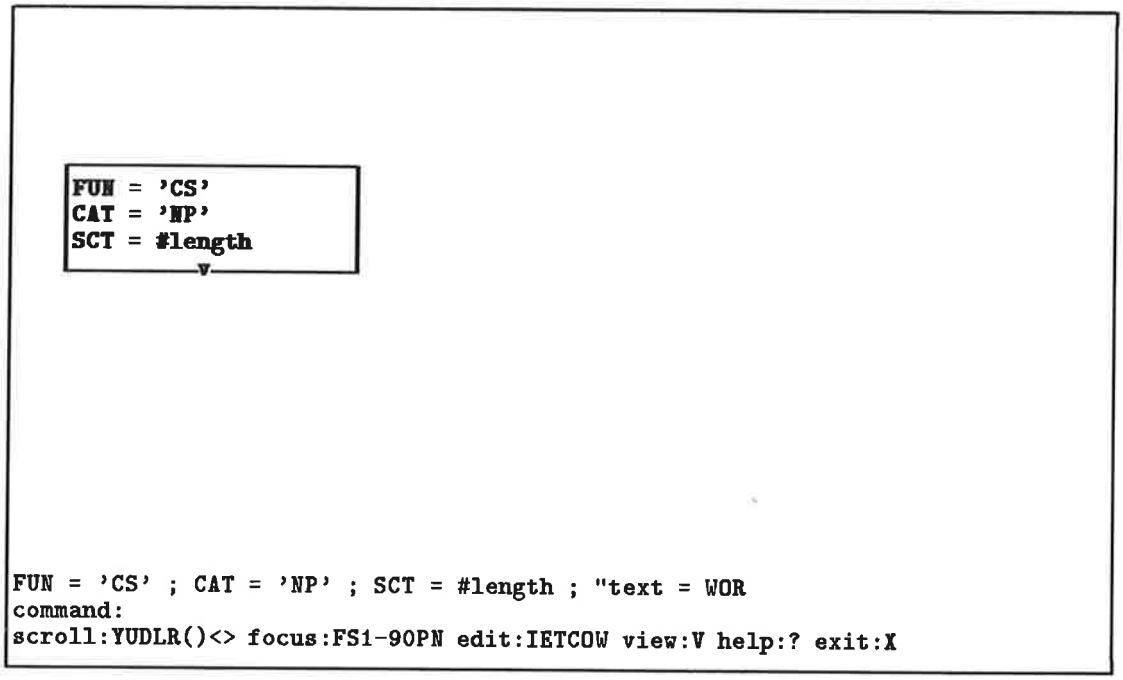

Figure 7a.

Example 7: activity variables.

A new kind of variables is introduced, which do not change with every match but are explicitly given a value.

The search pattern in figure 7a consists of single node, containing four specifications. However, there is only room for three of them in the box shown on the screen. Therefore, the lower border of the box is marked with an arrow ('v') to indicate that there is more than can be shown. In this case the fourth specification can be seen elsewhere on the screen, namely on the third line from the bottom where the contents of the focus are given.

Apart from the abbreviation NP (noun phrase) and the specifier SCT (son count), which stands for the number of immediate constituents, there seems to be nothing new here. The novelty comes to light when the activities are examined. The corpus intro consists of the action

```
#length := 1
```

and the match effect of

```
SCREEN( NEWLINE, "text )
USER
#length +:= 1
```

The operator := in the corpus intro should be read as 'becomes the value', so that after the corpus intro the variable #length is equal to the value one. As it is explicitly given a value in an activity, #length can not change value spontaneously during the matching process, but will have to stick to its assigned value. This means that in the first match the matching constituent will be a subject complement noun phrase with exactly one immediate constituent.

However, after the user has again handed back control to the program (i.e. after USER in the match effect) there is a change in value for #length. The operator +:= is a combination of adding values and assigning the resulting value to the variable, so that its net result is the incrementation by one of #length. In the next match there will have to be two immediate constituents.

On successive matches the exploration scheme will find subject complement noun phrases with an ever-increasing number of immediate constituents. Each time it finds one, it will show the corresponding words on the screen and stop.

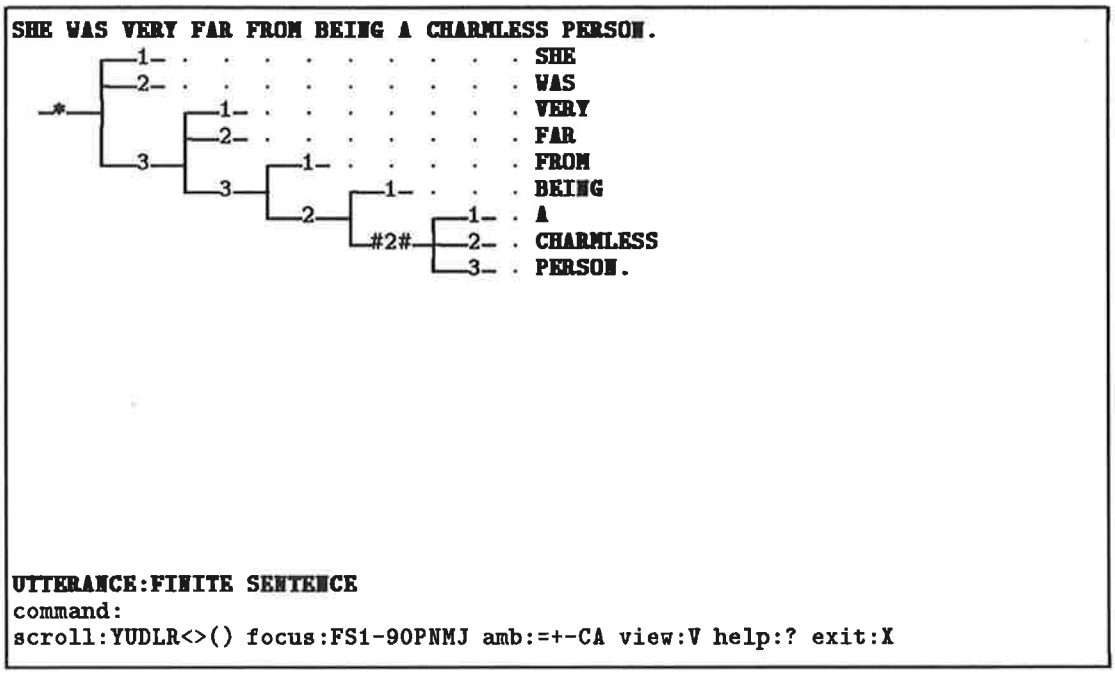

Figure 7b.

The third time the exploration scheme stops the matching utterance is the one in figure 7b. The subject complement is A CHARMLESS PERSON. Note that VERY FAR FROM BEING A CHARMLESS PERSON also is a subject complement with three immediate constituents. It will not be found as a match, however, because it is not a noun phrase.

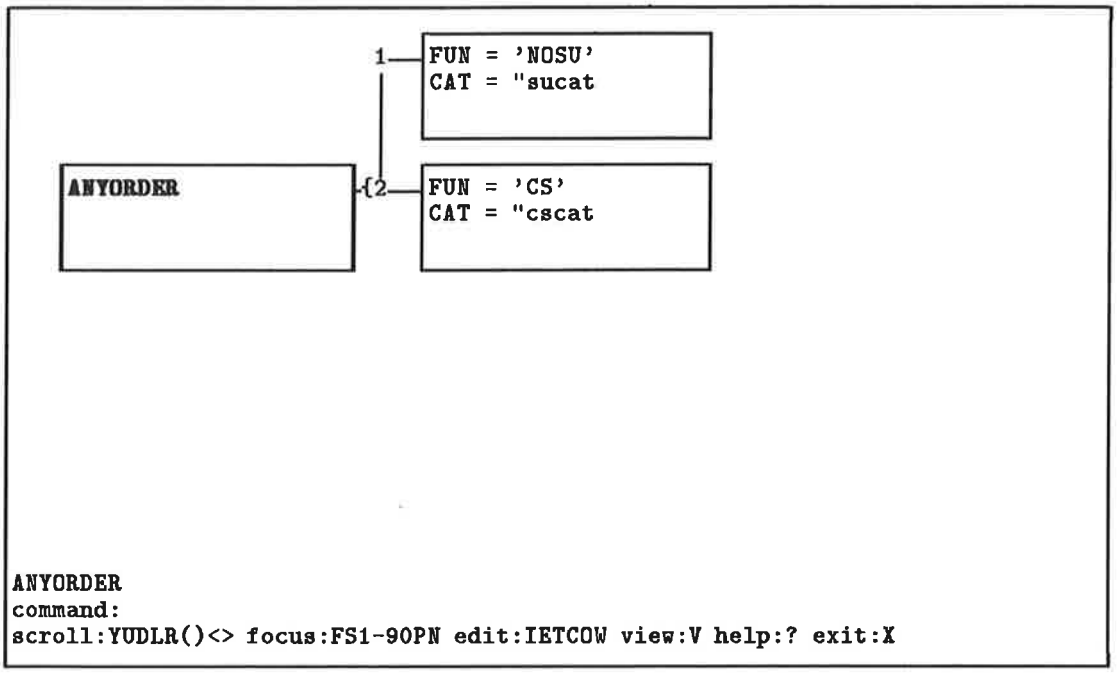

Figure 8a.

Example 8: frequency tables.

Another type of declaration is introduced, but the main purpose of this example is to show the means of producing frequency tables.

The pattern in figure 8a has one novel feature. It is the new declaration **ANYORDER**. Just as **ANYDEPTH** lifts the restriction on the length of connections, **ANYORDER** states that the order of the matching constituents is not necessarily the order given in the pattern. Here this means that the subject complement may well precede the notional subject. Again the presence of a declaration is indicated in the scheme editor, this time by the curly bracket '{'.

The corpus intro reads

```
$"catpairs := 0
```

and brings us yet another type of variable. A *string table* consists of not only one string but an entire list of them. Furthermore, each one is accompanied by a numerical value. The assignment of the value zero initializes `$"catpairs` so as to be an empty table.

Its use becomes clear when the match effect is examined:

```
"sucat + '-' + "cscat OF $"catpairs +:= 1
```

The variables `"sucat` and `"cscat`, representing string values stemming from the match of the pattern are concatenated (string addition is actually concatenation), separated by a hyphen. The resulting string is then viewed as an entry in the table `$"catpairs`, which is indicated by the operator `OF`. Finally, the numerical value associated with this entry is incremented by one. The net effect is that during the run of the exploration scheme the occurrences of all such pairs of categories are counted.

When the run is finished the corpus effect

```
SCREEN( TBL( $"catpairs ))
```

is executed. This will show the resulting table on the screen, formatted as a frequency table, i.e. the entries arranged vertically with the associated values shown alongside.

The exploration scheme goes through the corpus looking for notional subject with an associated subject complement either preceding or following it. Along the way it constructs a frequency table of the pairs of categories occurring in such a configuration. When finished, the resulting table is shown and control handed back to the user.

When the exploration scheme has gone through part of the corpus (only 20.000 words of text were used

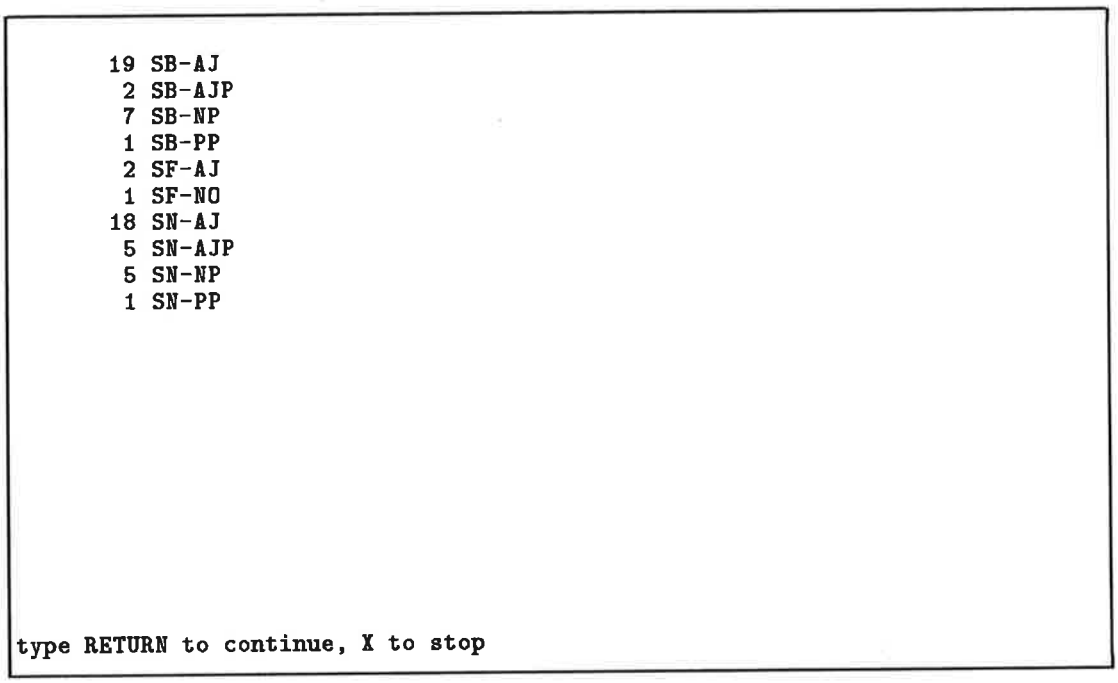

Figure 8b.

for this specific run) the screen of figure 8b appears. It shows the frequency table of the above-described category pairs. It seems that only clauses appear as notional subjects (SB = subordinated clause, SF = finite sentence and SN = nonfinite sentence). As subject complements are found adjectives (AJ), adjective phrases (AJP), noun phrases (NP), one single noun (NO) and prepositional phrases (PP).

Note that the corpus effect has the entire screen at its disposal, as opposed to the other activities, which are limited to one line for their messages. As the bottom line indicates this screen will be left when RETURN or X are typed. In this specific scheme, both will have the same effect as there is only one screen of output. However, if there would have been more than one screen pressing RETURN would go on to the next screen, while pressing X would skip the rest of the output and go back to the menu.

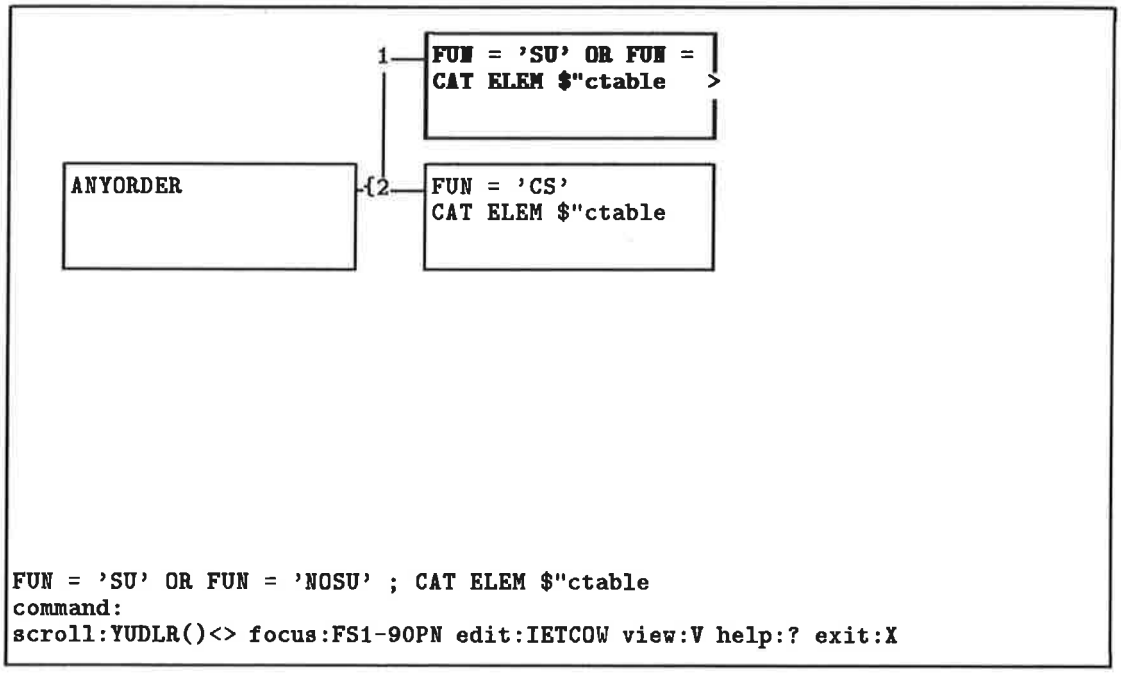

Figure 9a.

Example 9: tables revisited.

Two methods of providing choices between several matching constructs are demonstrated. One of these is by a new use of the tables introduced in example 8.

The pattern of the exploration scheme is shown in figure 9a. On the representational level there is a new marker for box overflow, this time at the right-hand side, namely the arrow pointing to the right ('>'). Again the focus has been moved to the overflowing node, so that the third line from the bottom of the screen shows the full contents of the node. There are also commands to scroll the box contents so it would be possible to view the extra information within its box.

Within the actual scheme the first new item is the use of the operator `OR`. It leaves some freedom of choice for the specification `FUN = 'SU' OR FUN = 'NOSU'`. The specification will be satisfied when the function label is either subject or notional subject. Note that all specifications up to now were just special cases. In fact any expression with numbers, strings, arithmetical operators, string operators and logical operators which in the end results in either `TRUE` or `FALSE` is valid as a specification.

A second way of providing choice in the labels needed is by specifying that they must be one of the entries in a table. As the numerical values in the table are not of actual importance, and the table is used as if it were a set, the operator used for this is `ELEM` ('is an element of'). The list of categories to choose from is provided in the corpus intro

```
$"ctable := STBL( 'SB', 'SF', 'SN' )
```

where `STBL` stands for string table. There are of course also numerical tables. In the case at hand the table with possible values for the categories is fixed throughout the entire corpus, but it would be possible to add or delete entries during the run of the exploration scheme. One could eg. collect an example of each occurring category of subject complement by noting down each category already found in a table and specifying in the pattern that the category in new matches may not be an entry in this table.

This last exploration scheme looks for utterances in which a subject complement complements either a normal or a notional subject. The order of the two is irrelevant but both must be clausal. When a match is found the scheme stops.

An utterance found by the scheme is shown in figure 9b. The subject is the nonfinite sentence `TO IMAGINE A THING`. The subject complement is also a nonfinite sentence, `TO GUARD AGAINST ITS REALLY HAPPENING`. As usual the matching nodes are marked.

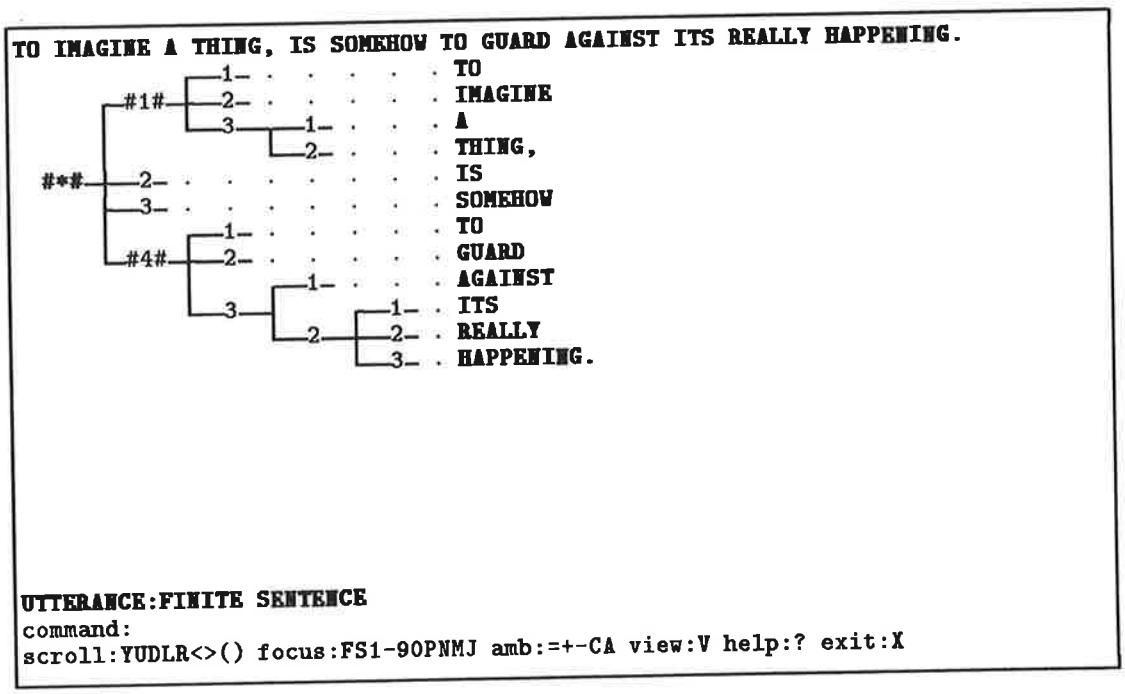

Figure 9b.

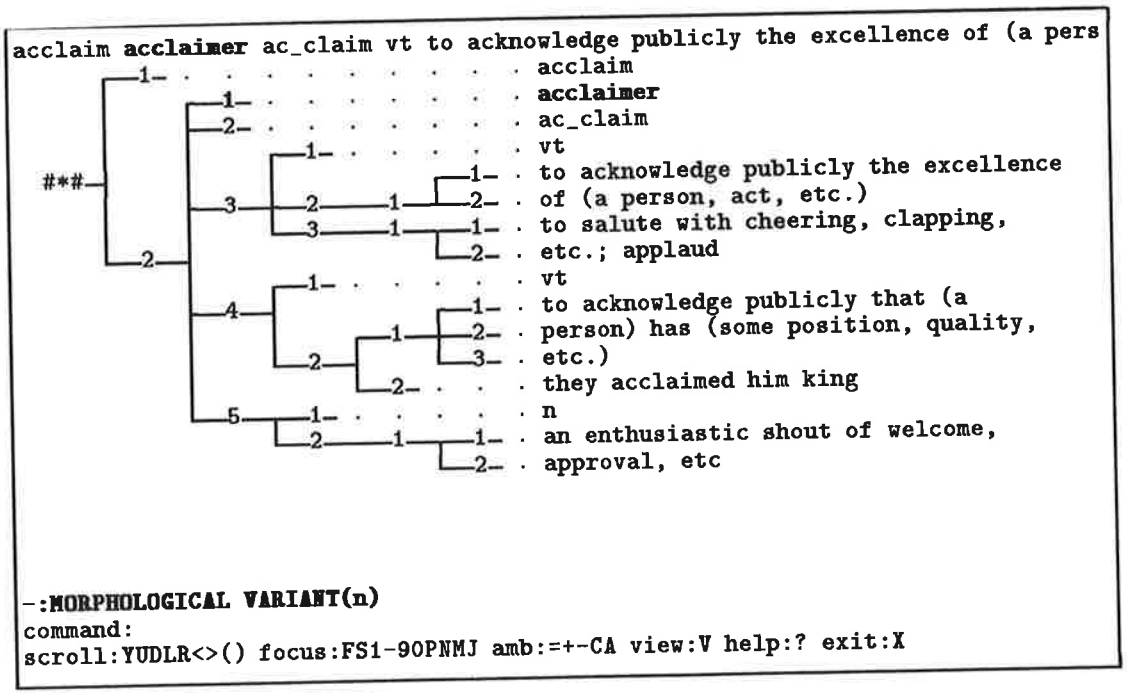

Figure 10.

Conclusion.

The examples given above present an overview of all but the more esoteric features of exploration schemes. They also give an impression of the richness of information present in syntactically analysed corpora. They cannot, due to the medium of the presentation, show the interactive possibilities, such as the navigation in the tree viewer and the creation and adaptation of an exploration scheme. The only way of really getting to know the system is trying it out.

Another important point which was not brought forward in the examples is the independence of the

program and the data. Not only English texts analysed in the way of the CCPP analyses can be stored in the *Linguistic DataBase*. As mentioned in the introduction, we are working on Spanish and Arabic at this very moment. For the time being, the Arabic is presented in a transliteration, but the facilities for use of non-roman fonts in the *LDB* are in an advanced stage of preparation. Furthermore, not only syntactic analysis trees can be stored in the *LDB*, but any kind of tree structure. An example of this is figure 10, which shows a dictionary entry in *LDB* format. The data was obtained by transforming the Collins Prolog Fact Base created in the CODER project at Virginia Tech ([4]). This fact base is itself a transformation of the Collins Dictionary and is available from the Oxford Archives. Note, in the labeling of the focus, that in these trees there is no function label but there are some attribute labels (at the focus the attribute 'n' indicates the nominal type of the variant).

The *Linguistic DataBase* is a program of about 25.000 lines, written in the language CDL2 ([10]). It is currently running on VAX, IBM PC/AT and SUN. For the SUN version, use was made of intermediary code in the C programming language. This means that versions for other UNIX based machines will be be easy to implement. If you have one of the machines mentioned above, you can get a copy of the *LDB* software, together with the Nijmegen corpus shown in the examples. For academic institutions only the costs of medium and postage are charged. If you have another machine and want to have this package or if you want to store data of your own in this database, please contact us. We can then see how we can help you.

All applications for the *LDB* or requests for further information can be sent to:

TOSCA Work Group
Dept. of English
University of Nijmegen
P.O.Box 9103
6500 HD Nijmegen
The Netherlands

Bibliography.

[1] Aarts, J. and Th. van den Heuvel (1985): 'Computational Tools for the Syntactic Analysis of Corpora', in: *Linguistics* 23.

[2] Aarts, J. and W. Meijs, eds. (1986): *Corpus Linguistics II. New Studies in the Analysis and Exploitation of Computer Corpora,* Amsterdam: Rodopi.

[3] Ditters, E. (1987): 'Progress Report ASCAMSA', in: Ditters, E. and N. Oostdijk, eds. (1987), Processing Arabic. report No.2, Nijmegen: TCMO, University of Nijmegen.

[4] Fox, E. A. et al. (1986): Building the CODER Lexicon, The Collins English Dictionary and its Adverb Definitions, Technical Report 86-23, Blacksburg: Dept. of Computer Science, Virginia Tech (VPI&SU).

[5] Hallebeek, J. (1987): 'Hacia un sistema de análisis sintáctico automatizado: el proyecto ASATE', in: Vide, C. ed. (1987): *Actas del II Congreso de Lenguajes Naturales y Lenguajes Formales,* Barcelona: Universidad de Barcelona.

[6] Halteren, H. van and Th. van den Heuvel (to appear): *Linguistic Exploitation of Syntactic Databases,* Amsterdam: Rodopi.

[7] Halteren, H. van and N. Oostdijk (1988): 'Using an Analysed Corpus as a Linguistic Database', in: Roper, J. ed. (1988): *Computers in Literary and Linguistic Computing,* (Proceedings of the XIIIth ALLC Conference, held at Norwich, 1986).

[8] Heuvel, Th. van den (1987): 'Interaction in Syntactic Corpus Analysis', in: Meijs, W., ed. (1987): *Corpus Linguistics and Beyond,* Amsterdam: Rodopi.

[9] Keulen, F. (1986): 'Some Experiences with a Semi-Automatic Analysis of Contemporary English' in: Aarts and Meijs, eds. (1986).

[10] Koster, C. H. A. (1977): *CDL - A Compiler Implementation Language, Methods of Algorithmic Language Implementation,* Lecture notes in Computer Science 47, Berlin: Springer.

[11] Oostdijk, N. (1986): 'Coordination and Gapping in Corpus Analysis', in: Aarts and Meijs, eds. (1986).

[12] Oostdijk, N. (to appear, 1989): 'Ambiguity in Syntactic Corpus Analysis', in: Proceedings of the Fifteenth International Conference on Literary and Linguistic Conputing, Jerusalem June 5-13, 1988.