Guide to the Unification Process and its Implementation. Progress Report on Extending the Grammar.

Cecile L. Paris and TjoeLiong Kwee[1]

Department of Computer Science

Columbia University

New York City, New York, 10027 USA

CUCS-208-85

Abstract

There are two steps in generating an English sentence using a functional grammar: unification with the grammar and linearization of the resulting structure. This report is intended as an introduction and a guide to the unification process and its implementation. The program and grammar used are modified versions of those used in the TEXT system [McKeown 82]. We describe the major functions of the program, noting how we have made it more general and more efficient. We also describe the extensions and improvements made to the original functional grammar used in the TEXT system.

NOTE: This report is primarily aimed at persons interested in using this particular implementation. A general introduction to the unification process and use of functional grammars can be found in [Kay 79] and [McKeown 82].

September 1985

---

[1]Visiting from the University of Amsterdam, Institute for General Linguistics; Spuistraat 210, telefoon 525; Postbus 19188, 1000 GD Amsterdam, The Netherlands

Table of Contents

This report is intended as an introduction and a guide to the unification process and its implementation. The program we use is a modified version of Steve Bossie's [McKeown 82]. We describe the major functions in the program, noting modifications we have made. Note that this report does not duplicate information presented in [Kay 79] and [McKeown 82].

The appendices contain the following:

- A simple grammar used in the examples.

- A glossary of symbols used in the unifier.

- A list of files left by TjoeLiong Kwee with a description of their contents.

- A trace of the unification process of a simple input and the grammar shown in Appendix I. The input used in this trace is a completely specified input.

- A trace of the unification process of another input and the grammar shown in Appendix I. The input is simplified (not completely specified) to demonstrate of the handling of unattached constituents).

- An example of one of the problems encountered.

- How to use the unifier.

1. General Introduction to the Unification process

There are two steps in generating an English sentence: ***unification*** with the grammar, and ***linearization*** of the resulting structure. The input to the unification process is a deep structure of the sentence to be generated (see [McKeown 82]). It does not contain all the syntactic information necessary to generate the sentence. The unifier unifies the input with the grammar to enrich the input with all the necessary syntactic information. The linearizer then takes this enriched input and produces a flat list, the English sentence. The grammar is a ***functional grammar***, as described by Kay, and the input is in the same form as the grammar. Introductions to functional grammars and the unification process can be found in [Kay 79] and [McKeown 82].

Example using the simple grammar of Appendix I:

Input:

```
((cat S) (verb ((v == compress)))
         (prot ((cat np) ((article ((lex def)))
                          (nnp ((n == diaphragm))))))
         (goal ((cat np) ((number plur)
                          (article ((lex def)))
                          (nnp ((adj == small)
                                (nnp ((n == granule)))))))))
```

Note that the input does not specify the verb voice, the number agreement of the verb, etc.
Such information will be added by the unification.
The input only indicates the action, the actor (i.e. the *prot*), and the object of the action (i.e. the *goal*).

Enriched structure after unifying this input with the grammar:

```
((((GOAL ((NNP ((NNP ((N ((LEX GRANULE) (CAT NOUN))) (CSET (N))
                         (PATTERN (N POUND)) (TAIL NONE) (PP NONE)
                         (ADJ NONE) (CAT NNP)))
                   (ADJ ((LEX SMALL) (CAT ADJ))) (CSET (ADJ NNP))
                   (PATTERN (ADJ NNP)) (CAT NNP)))
             (ARTICLE ((LEX DEF) (CAT ARTICLE)))
             (CSET (ARTICLE NNP))
             (PATTERN (DOTS ARTICLE NNP DOTS)) (CAT NP)
             (NUMBER PLUR)))
    (VERB ((V ((LEX COMPRESS) (CAT VERB))) (CSET (V)) (PP NONE)
           (PATTERN (V DOTS)) (VOICE ACTIVE) (NUMBER NONE)
           (CAT VERB-GROUP)))
    (PROT ((NNP ((N ((LEX DIAPHRAGM) (CAT NOUN))) (CSET (N))
                 (PATTERN (N POUND)) (TAIL NONE) (PP NONE)
                 (ADJ NONE) (CAT NNP)))
           (ARTICLE ((LEX DEF) (CAT ARTICLE)))
           (CSET (ARTICLE NNP))
           (PATTERN (DOTS ARTICLE NNP DOTS)) (CAT NP)))
    (CSET (PROT VERB GOAL)) (PATTERN (PROT VERB GOAL POUND))
    (CAT S))
   ((TAIL NONE) (PP NONE) (ADJ NONE) (NUMBER PLUR))))
```

Output after linearization:

(THE DIAPHRAGM COMPRESSES THE SMALL GRANULES!.)[1]

The unification process unifies each ***functional description*** in the input with the grammar (or with the appropriate sub-part of the grammar).

To unify a functional description, a unifier considers each ***attribute-value pair*** in the

[1] All punctuation marks are special characters in PSL. They are thus printed with a preceeding exclamation mark. Note furthermore the parenthesis: the output of the unifier is a list containing the generated sentence.

grammar and unifies its value part with that of the corresponding attribute in the input if present.

If an attribute occurs in the grammar but not in the input, the input is enriched with the attribute-value pair from the grammar. (That is, the resulting structure is the ***union*** of the input and the grammar.) The grammar is thus used to enrich the input with all the syntactic information necessary to produce a sentence. The value ***any***, which may appear in the grammar, is treated as a special case: if ANY is specified, the unification will succeed ***if and only if*** the corresponding attribute is present in the input.

If the input and the grammar have the attribute in common, then:

- If the value in the grammar is ANY, it automatically matches the other value, whatever it may be, ***provided that there is one***.

- If either value is a symbol, the unification succeeds when the values are equal.

- If both values are functional descriptions, they are unified in turn and success depends on this further unification.

- Unification fails in all other cases.

If the grammar has alternatives, these are all tried in turn. Our unifier returns either the first or all successful results, depending on the value of the variable *one-result* (True or False).

A functional description may contain one or more constituents that need to be unified in turn. In our input example above, the first functional description to be unified with the grammar will be (CAT S). Once (CAT S) is unified, its constituents **verb** , **prot** and **goal** have to be unified.

The following is a summary of the algorithm used to unify one functional description from the input with the grammar:

```
(a) Choose the appropriate sub-grammar from the grammar.
    Repeat (i) through (iii) until EITHER there are no
    attribute-value pairs in the grammar OR unification of
    two pairs fails.
            i.  - Take the next attribute-value pair from the grammar
            ii. - Look for a corresponding attribute in the input
            iii.- If there is one: unify the two values
                  If there is none: enrich the input with attribute-
                    value pair from the sub-grammar
(b) If (a) failed, return.
    Otherwise, if there are any constituents left in the input,
    repeat this unification process recursively for each constituent.
```

2. Trace of the unification process

This traces the unification of a simple input with the grammar given in Appendix I. This trace illustrates only the unification process and not the function calls[2]. For simplicity only the beginning of the trace is shown.

```
Input:
(SETQ X2
      '((CAT NP) (ARTICLE == DEF)
        (NNP ((ADJ == SMALL)
              (NNP ((ADJ == NEW)
                    (NNP ((ADJ == LIGHT) (NNP ((N == GRANULE)))))))))))
```

Input after UNIFY has enriched the lexical items[3]:

Let I be the input and G the grammar with which the input will be unified. I and G are arguments to the unifier.

```
(SETQ I
  '((CAT NP) (ARTICLE ((LEX DEF)))
    (NNP ((ADJ ((LEX SMALL)))
          (NNP ((ADJ ((LEX NEW)))
                (NNP ((ADJ ((LEX LIGHT)))
                      (NNP ((N ((LEX GRANULE)))))))))))
```

G = the whole grammar, as found in Appendix I.[4]

**The ''top-level'' ALT in the grammar (see Appendix I) indicates there are alternatives in the grammar.
The appropriate alternative is chosen based on CAT in the input, if possible. In this case, (CAT NP) is the appropriate alternative.**

The unification process starts now with this category as the sub-grammar:

```
(SETQ G
      '((ALT (((ARTICLE NONE) (PATTERN (DOTS NNP DOTS)))
              ((ARTICLE ANY) (ARTICLE ((CAT ARTICLE) (LEX ANY)))
               (PATTERN (DOTS ARTICLE NNP DOTS)))))
        (NNP ((CAT NNP)))))
```

**G has alternatives again.
It is not possible anymore to choose the appropriate alternative based on the category indicated in the input. So each alternative is tried in turn:**

First alternative:

```
(SETQ G '((ARTICLE NONE) (PATTERN (DOTS NNP DOTS)) (NNP ((CAT NNP)))))
```

Second alternative:

```
(SETQ G
      '((ARTICLE ANY) (ARTICLE ((CAT ARTICLE) (LEX ANY)))
        (PATTERN (DOTS ARTICLE NNP DOTS)) (NNP ((CAT NNP)))))
```

---

[2] A complete trace for the same example is shown in Appendix VI

[3] See the explanation of the top-level function **unify** in the next section.

[4] This is a short hand for (SETQ G '((alt (((cat S) (alt (...).

1- First alternative:

(SETQ G '((ARTICLE NONE) (PATTERN (DOTS NNP DOTS)) (NNP ((CAT NNP)))))

******* first Attribute-value pair from the grammar: (ARTICLE NONE)

Now, we try to unify the input with this pair.

An attribute-value pair with the same attribute is found in the input:
(ARTICLE ((LEX DEF)))

The value from the input and that from the grammar are unified.
The unification process starts again:
I = ((LEX DEF))
G = NONE

Unification fails, as ''NONE'' can be unified successfully only with ''NIL''

--> Alternative fails

2- Next alternative:

(SETQ G
'((ARTICLE ANY) (ARTICLE ((CAT ARTICLE) (LEX ANY)))
(PATTERN (DOTS ARTICLE NNP DOTS)) (NNP ((CAT NNP)))))

******* First Attribute-value pair from the grammar: (ARTICLE ANY)

The input is unified with this pair:

Attribute-value pair with corresponding attribute found in the input:
(ARTICLE ((LEX DEF)))

The 2 values are unified:

I = ((LEX DEF))
G = ANY

Unification succeeds...

****** Next Attribute-value pair from the grammar:
(ARTICLE ((CAT ARTICLE) (LEX ANY)))

The iput is unified with this pair:

Attribute-value pair found in the input: (ARTICLE ((LEX DEF)))

The 2 values are unified:

I = ((LEX DEF))
G = ((CAT ARTICLE) (LEX ANY)))

G is itself a *functional description*.
Unification starts again:

** First attribute value pair: (CAT ARTICLE)
It is unified with the input I = ((LEX DEF))

No pair with attribute ARTICLE is found in the input.
The input is enriched with the attribute-value pair
from the grammar.

Enriched input:
I = ((CAT ARTICLE) (LEX DEF))

```
        This enriched input is now unified with  the next
        attribute value pair from the  grammar

     ** Next attribute value pair: ((LEX ANY))

        Pair found in the input with attribute LEX: (LEX DEF)

        The 2 value are unified: DEF and ANY
        Unification succeeds.

Unification with ((CAT ARTICLE) (LEX ANY)) succeeds;
The input has already been enriched with (CAT ARTICLE)

The new (enriched) input is:

(SETQ I
  ((ARTICLE ((LEX DEF) (CAT ARTICLE)))
   (CAT NP) (NNP ((ADJ ((LEX SMALL)))
                  (NNP ((ADJ ((LEX NEW)))
                        (NNP ((ADJ ((LEX LIGHT)))
                              (NNP ((N ((LEX GRANULE))))))))))))

******* Next Attribute-value pair from the grammar:
        (PATTERN (DOTS ARTICLE NNP DOTS))

The pattern is added to the input.

New input:

(SETQ I
 '((PATTERN (DOTS ARTICLE NNP DOTS))
   (ARTICLE ((LEX DEF) (CAT ARTICLE)))
   (CAT NP)
   (NNP ((ADJ ((LEX SMALL)))
         (NNP ((ADJ ((LEX NEW)))
               (NNP ((ADJ ((LEX LIGHT)))
                     (NNP ((N ((LEX GRANULE))))))))))))

******* Next Attribute-value pair from the grammar:
              (NNP ((CAT NNP)))

Pair from the input with the corresponding attribute:
    (NNP ((ADJ ((LEX SMALL)))
          (NNP ((ADJ ((LEX NEW)))
                (NNP ((ADJ ((LEX LIGHT)))
                      (NNP ((N ((LEX GRANULE))))))))))

The  2 values are unified:
     I = ((ADJ ((LEX SMALL)))
          (NNP ((ADJ ((LEX NEW)))
                (NNP ((ADJ ((LEX LIGHT)))
                      (NNP ((N ((LEX GRANULE))))))))))
     G = ((CAT NNP)))

 As there is no attribute value pair in the input with
 attribute 'CAT, the input is enriched with (CAT NNP):

 New Input:

 I = ((NNP ((CAT NNP)
            (ADJ ((LEX SMALL)))
            (NNP ((ADJ ((LEX NEW)))
                  (NNP ((ADJ ((LEX LIGHT)))
                        (NNP ((N ((LEX GRANULE))))))))))
```

```
          (PATTERN (DOTS ARTICLE NNP DOTS))
          (ARTICLE ((LEX DEF) (CAT ARTICLE)))
          (CAT NP))
```

(SETQ G 'NIL)

Alternative succeeded

****
The top level constituent (cat NP) is processed.
****

Now the inside constituents are processed: 1) ARTICLE and 2) NNP

1) ARTICLE constituent

I = ((LEX DEF) (CAT ARTICLE))

G = the whole grammar

Again the grammar has alternatives.
The appropriate alternative is chosen based on CAT in the input:
(CAT ARTICLE)
Unification proceeds with this category as the sub-grammar:

G = ((ALT (((LEX INDEF)) ((LEX DEF)))))

There are alternatives, but we are at the lexical level. The alternative with the correct lexical choice is taken.

G = ((LEX DEF))

Alternative succeeds...

****

2) NNP constituent:

```
I = ((CAT NNP) (ADJ ((LEX SMALL)))
               (NNP ((ADJ ((LEX NEW)))
                     (NNP ((ADJ ((LEX LIGHT)))
                           (NNP ((N ((LEX GRANULE)))))))))))
```

G = the whole grammar

The grammar has alternatives.
The appropriate alternative is chosen based on CAT in the input:
(CAT NNP)
The input is unified with the sub-grammar corresponding to this category:

```
I =  ((CAT NNP) (ADJ ((LEX SMALL)))
      (NNP ((ADJ ((LEX NEW)))
            (NNP ((ADJ ((LEX LIGHT)))
                  (NNP ((N ((LEX GRANULE)))))))))))

G =  ((ALT (((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
                        (N ((CAT NOUN) (LEX ANY))))
           ((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY)))
                             (PATTERN (ADJ NNP)))
                  ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
            (NNP ((CAT NNP)))))))
```

1- First alternative:

```
G  = ((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
                 (N ((CAT NOUN) (LEX ANY))))
```

******* First pair from the grammar: (ADJ NONE)

Corresponding pair in the input: (ADJ ((LEX SMALL)))

The 2 values, SMALL and NONE are unified. Unification fails.

Alternative has failed

2- Next alternative:

```
G = ((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY))) (PATTERN (ADJ NNP)))
           ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
    (NNP ((CAT NNP))))
```

There are yet more alternatives.

2.1first alternative:

```
G = ((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY))) (PATTERN (ADJ NNP))
     (NNP ((CAT NNP))))
```

******* First Attribute-value pair from the grammar: (ADJ ANY)

Corresponding pair found in the input: (ADJ ((LEX SMALL)))

The 2 values, SMALL and ANY are unified. Unification succeeds.

******* Next pair from the grammar: (ADJ ((CAT ADJ) (LEX ANY)))

Corresponding pair found in the input: (ADJ ((LEX SMALL)))

The 2 values are unified:

```
I = ((LEX SMALL))
G = ((CAT ADJ) (LEX ANY)))
```

Here again, the grammar value is itself a functional description. The unification process is repeated recursively:

First, (CAT ADJ): Since there is no attribute 'CAT in the input, the input is enriched with (CAT ADJ).

Then, (LEX ANY): The process results in unifying the two values SMALL and ANY. Unification succeeds.

So we now have the following enriched input:

```
I =   (ADJ ((LEX SMALL) (CAT ADJ)))
```

******* Next pair from the grammar: (PATTERN (ADJ NNP))

The new input is:

```
I = ((ADJ ((LEX SMALL) (CAT ADJ))) (CAT NNP)
     (NNP ((ADJ ((LEX NEW)))
           (NNP ((ADJ ((LEX LIGHT)))
                 (NNP ((N ((LEX GRANULE)))))))))
```

This pattern is just added to the input:

New input:

```
I = ((PATTERN (ADJ NNP)) (ADJ ((LEX SMALL) (CAT ADJ)))
     (CAT NNP)
     (NNP ((ADJ ((LEX NEW)))
           (NNP ((ADJ ((LEX LIGHT)))
                 (NNP ((N ((LEX GRANULE)))))))))

******* Next pair from the grammar: (NNP ((CAT NNP)))

Corresponding pair found in the input:
    (NNP ((ADJ ((LEX NEW)))
          (NNP ((ADJ ((LEX LIGHT)))
                (NNP ((N ((LEX GRANULE)))))))))

  The 2 values are unified....

[ The unification process continues... Its result
  is given to the linearizer which returns either a
  list containg the English translation of the original
  input or nil. ]

(THE SMALL NEW LIGHT GRANULE!.)
```

3. Explanations of the main functions used by the unifier

What follows is the explanation of the main functions used in the unifier. To simplify it, I will omit for now the two features ***up-arrow*** and ***loose*** which will be introduced later.

3.1. Unify

Unify is the top-level function of the unifier. Given an input and a grammar, it will return an English sentence if the unification process succeeds, NIL otherwise.

Unify first checks that all the words used in the input are defined in the dictionary. It also completes the lexical definitions in the input:

Example:

In the input, the lexical items are given as:

(n == granule), (v == compress),...

This notation is short-hand notation for:

(n ((lex granule))), (v ((lex compress))), ...

Unify preprocesses the input to check that the words are in the lexicon and adds this complete notation for the lexical items.

Unify then calls the function which does the unification proper: **uavs-ls.** This function is called on the top-level (higher) functional description of the input (i.e NP in our previous example). Upon return, **unify** calls **ucon-many** if there are any other constituents in the input that need to be unified. **Ucon-many** unifies each one in turn.

Finally, **unify** calls the linearizer to produce a flat structure, the English sentence.

3.2. Uavs-ls

Uavs-ls[5] is given the input to unify *(i)*, the grammar to unify the input with *(g)*, and the stack of all the functional descriptions that the current input is embedded in, *(fd-stack)*. **Uavs-ls** is the function which does the actual unification. It is called whenever two values need to be unified:

- For a functional description and a sub-grammar, **uavs-ls** is called for each attribute value pair of the sub-grammar

- When a constituent needs to be unified, **uavs-ls** is first called with the whole grammar

- When a value given as input to **uavs-ls** is itself a functional description, **uavs-ls** is called recursively (through other functions).

1) If the grammar is null (there are no more attribute-value pairs left), the input is returned.

2) At the lowest level, two symbols are unified by calling the function **sym-h** (the symbols handler).

For example, the symbols:

```
''np'' with ''np''                          --> success
''np'' with ''any'' (the wild card) --> success
                                            (remember that ''any''
                                             succeeds with any non-null
                                             symbols)
''nil'' with ''np''                         --> failure (nil)
''np'' with ''pp''                          --> failure (nil)
''none'' with ''np''                        --> failure (nil)
''none'' with ''nil''                       --> success
```

3) At higher levels, **uavs-ls** takes each attribute-value pair of the grammar and acts according to the name of the attribute:

- *Alt*: The grammar has alternatives. The function **alt-h-ls** calls **uavs-ls** on each alternative.

- *Pattern*: A pattern indicates to the linearizer the order in which words should appear. The function **pattern-h-ls** adds the pattern to the input.

---

[5]**Uavs-ls**: Unify one attribute-value pair. *Ls* means that *loose* is passed to the function.

- *Cset*: A functional description may contain constituents that need to be unified in turn. The constituent set *cset* has the list of these *"inside constituents"*. It is used to determine whether **uavs-ls** should be called recursively on inside constituents. The constituent set is added in the input by the function **cset-h-ls**.

- *All other attributes*: Given the attribute-value pair of the grammar, the program tries to find in the input a pair with the corresponding attribute. If there is one, the two values will be unified. If there is no such attribute in the input, the attribute-value pair from the grammar will be added to the input. The function **av-h-ls** handles these attributes.

3.3. Alt-h-ls

Alt-h-ls[6] is called when there are alternatives in the grammar. **Uavs-ls** is called with each alternative in turn until a success is found[7].

When separating the different alternatives, care has to be taken to make sure the sub-grammar is complete.

Example:

```
G = ((alt ((prot none))
          ((prot any)))
   (goal any) ...)
```

The first alternative to try is sub-g1 = ((prot none) (goal any))

The second: sub-g2 = ((prot any) (goal any))

--> We have to make sure that the sub-grammar contains whatever pairs followed the ALT.

These following details are not a fundamental part of the unification algorithm but they are important for efficiency reasons. As the unification process is a lengthy one, we tried to make it as efficient as possible.

3.3.1. Category attribute (CAT)

Normally, all the alternatives are tried in turn. However, in some cases, the input includes a clue about which alternative to choose. This is often the case for the CAT attribute.

As an example, in the input given in the trace, (CAT NP) indicates that this input is of category NP (a noun phrase).

The grammar is a long functional description, made up of several alternatives, one for each functional constituent possible:
S for a sentence, NP for a noun phrase, PP for a prepositional phrase, etc.

When an input is unified with the whole grammar, the alternatives have to be tried until the one that corresponds to the input is found. In some cases however, the category is already indicated in the input: that is, the input has an attribute-value pair with the attribute CAT. In such case, instead of trying every possible

---

[6] **Alt-h-ls**: Alternatives handling

[7] As mentioned previously, a variable can be set so that all successful results are returned.

alternative, we take advantage of the clue given in the input to choose the appropriate sub-grammar immediately, and **uavs-ls** is called with this alternative To achieve this effect, **alt-h-ls** searches the input for the attribute-value pair with attribute CAT and calls **uavs-ls** with the alternative that contains the same category, if there is such an alternative.

Another speed up was obtained with the following observation: The sub-grammar which is passed to the recursive call to **uavs-ls** has, as its first attribute-value pair, (CAT xx). As we chose this sub-grammar because this pair was also present in the input, it is not useful to unify (CAT xx) from the input with (CAT xx) from the grammar using **uavs-ls** as would be done normally. We thus cut this pair out of the sub-grammar to call **uavs-ls**.

3.3.2. Lexicon

The lexicon is represented in the grammar as alternatives: one alternative per word. When a word from the input is unified with the grammar, all the alternatives are tried in turn until the correct word is found (that is the word equal to that in the input). Again, instead of calling **uavs-ls** on every lexical choice, alt-h-sl finds the appropriate word (and thus the appropriate alternative), after checking that unification reached the lexical level[8][9].

[8]The lexical level is reached when all the alternatives have the attribute LEX

[9]I made the two changes just described in June 1985

3.4. The Three Functions that Handle the Attribute-Value Pairs

1. **Av-h-ls** (Attribute-value pair handling)
2. **Cntrl-av-h-ls** (Central attribute-value pairs handling)
3. **Uavs-many-ls** (Unify many attribute-value pairs)

Av-h-ls is normally called with an attribute value pair from the grammar. The attribute is neither ***pattern***, ***cset***, or **alt** (for alternative). **Av-h-ls**, **Cntrl-av-h-ls** and **Uavs-many-ls** together unify all the attribute-value pairs of the grammar with the input:

Av-h-ls:

Av-h-ls finds the first attribute value pair from the grammar. The attribute is the ***path***. It calls **cntrl-av-h-ls** with this path and its value in the grammar, the input, and the remaining pairs in the grammar. **Cntrl-av-h-ls** finds a pair in the input with the same attribute (if there is one) and unifies the two values. Upon success, the unification proceeds by calling **Uavs-many-ls** to unify the input (possibly enriched) with the remaining pairs of the grammar.

Cntrl-av-h-ls:

Given an attribute from the grammar, this function searches the input for a pair with the same attribute.

- If it is found, the value from the grammar and that from the input are unified by calling **uavs-ls** with the input value as the input and the grammar value as the grammar. Note that the values can be symbols or functional descriptions.

- If it is not found, the input is enriched with the attribute-value pair from the grammar (unless the value from the grammar pair is the wild card ***any***)

[Note that in either case **uavs-ls** is called with the two values. If the attribute was not found, the input value is *NIL*. Thus , **uavs-ls** is called recursively twice for any attribute value pair from the grammar.]

After unifying the two values, **Cntrl-av-h-ls** calls **uavs-many-ls** with the remaining pairs of the grammar and the **possibly enriched input**.

Uavs-many-ls:

Given a list of attribute-value pairs, **uavs-many-ls** (*)[10] calls the appropriate function to unify the pairs one at a time.

[10] An asterisk indicates that there are details in the function that I do not fully understand. These details are also pointed out in the code.

3.5. Functions that handle the unification of constituent contained in the CSET

1. **Ucon-many(*)**
2. **Ucon**
3. **Ucon-proc(*)**

Once a functional description has been unified with the appropriate part of the grammar, the program looks for any constituent that remains to be unified. A non-null constituent set indicates that there are such constituents, and the program unifies them in turn. This process is also recursive, as, once a constituent has been unified, its CSET may indicate that there are yet more constituents to unify.

Ucon-many:

Ucon-many(*) is called when a functional description has been unified. It prepares the setting for the function that checks for a non-null constituent set, **ucon**.

Ucon:

Ucon searches the input for its constituent set(***cset***). If the ***cset*** is null, then, the unification is finished for that input and we are at the lexical level. Otherwise, **ucon-proc** is called to unify each constituent with the whole grammar.

Ucon-Proc:(*)

At this point, we know that the ***cset*** is not empty. **Ucon-proc** calls **uavs-ls** to unify each member of the ***cset*** with the complete grammar. This process is repeated if the resulting ***cset*** is non-null. Therefore, **ucon-proc** also calls **ucon-many**.

4. Special Feature: use of path

In some cases, a value for an attribute is not found in the current functional description (FD), but in the FD in which the current one is embedded. The simplest example is this phenomenon is found when the program is looking for the verb agreement[11].

When looking for the verb agreement, the current FD is the description concerning the VERB, and the FD in which it is embedded is S (the sentence level). The attribute value pair from the grammar is:

(verb (... (number (^ prot number)) ...))

This form means that the verb number should be the number of the protagonist.

The protagonist is not part of the current FD however. This is why the ***up-arrow*** is used: the ***up-arrow*** indicates that the value should be looked for in the FD in which the current one is embedded. In this example, the program will look into the "*S*" FD to find the number of the protagonist. That number is then returned as the value of the verb number[12].

There are also cases where, because of the recursive definition of some constituents (the noun phrase for example, in which each new adjectives adds one level), we cannot know ahead of time which of the 'higher' FD's contain the needed value. In such cases, ***^** is used in the grammar[13].

How the up-arrow is handled by our unifier

The function Av-h-ls

Given an attribute-value pair, **av-h-ls** calls **cntrl-av-h-ls** with the value from the grammar, the attribute and the input to unify. When the value in the grammar contain an up-arrow, as in the two pairs given below, the ***real*** value of the attribute needs to be found in the input:

Pairs in which the value is not explicitly given:

(attribute-name (↑ ...))

---

[11]For another introduction to ***paths***, see [McKeown 82].

[12]This is why the ***fd-stack*** is passed to the functions.

[13]see [McKeown 82] for a more detailed explanation of this case.

```
(attribute-name (*↑ ...))
```

In these cases, **av-h-ls** finds the real value before calling **cntrl-av-h-ls**. The function **get-val-if-path** finds the value by looking into the appropriate constituent.

5. Introduction to Loose:

The unifier program was designed to allow for simplified input. To illustrate why this is desirable, consider the following (simple) grammar description of a noun-phrase:

Grammar for a Noun Phrase in rule form:

```
NP --> article NNP
NNP --> adjective NNP / NOUN
```

Grammar for a Noun Phrase in the Functional Grammar formalism:

```
(alt
 (((cat NP) ((article any) (article ((cat article) (lex any)))
             (pattern (dots article nnp dots))
             (nnp ((cat nnp)))))

  ((cat NNP) ((alt (((adj none) (pattern (n pound))
                    (n ((cat noun) (lex any))))
                   ((adj any) (adj ((cat adj) (lex any)))
                    (pattern (adj nnp))
                    (nnp (cat nnp))))))))))
```

(Note that this definition is recursive)

Using this definition, the noun phrase "the new small light granule" is written as:

```
((cat np) ((article == def)
           (nnp ((adjective == new)
                 (nnp ((adjective == small)
                       (nnp ((adjective == light)
                             (nnp ((noun == granule))))))))))))
```

This example illustrates how complex the input can become. In complex sentences, having to write this complete specification of the input can be very tedious. We want the program to allow the same input to be:

```
((cat np) ((article == def) (adjective == new)
           (adjective == small)
           (adjective == light) (noun == granule)))
```

This input form is clearly simpler and more desirable than the previous one. However, in this form, the attachment of constituents is unspecified. The program is able to process these unattached constituents through the variable *loose*. In our example, the unification process will add 'NNP to the input structure where required,

Loose is a list of unattached constituents. As the unification proceeds, elements of *loose* are picked up and attached in the appropriate constituent.

The easiest way to explain how *loose* is used is with an example. Let us trace roughly the unification of the simplified input given above with the sample grammar given for the Noun Phrase:

1) **Uavs-ls** is called with the input and the sub-grammar **((cat np) ...)**.

2) The first attribute value pair is taken from the grammar:
(article any)

The input is searched for the attribute **article**. The pair is found, and its value is **(lex def)**. This value is unified with the value from the grammar **(lex any)**. Unification succeeds.

3) The next pair from the grammar is taken:
(article ((cat article) (lex any)))

Again, the input is searched for the attribute **article**. The input value is, as before, **(lex def)**. The value from the grammar is now **((cat article) (lex any))**. These two values are unified by calling **uavs-ls** again. At this point,

```
i = (lex def) and
g = ((cat article) (lex any)).
```

Since the input does not contain the attribute **cat**, it is enriched with the pair **(cat article)** as a result of the unification. Then, unifying **(lex any)** with **(lex def)** succeeds. After unification, the enriched input is:
(article ((cat article) (lex def))).

4) The next pair is taken from the grammar. The already enriched input is:

```
((cat np) (article ((cat article) (lex def)))
          (adj ((lex big))) (adj ((lex red))) (adj ((lex old)))
          (n ((lex car))))
```

The next pair is the **pattern**. It is added to the input.

5) The input is now as in (4) but enriched with the pattern. The next pair in the grammar is **(nnp ((cat nnp)))**. The input is unified with this pair. As there is no **nnp** attribute in the input, the pair is added to the input.

Resulting input:

```
((cat np) (article ((cat article) (lex def)))
          (adj ((lex big))) (adj ((lex red))) (adj ((lex old)))
          (n ((lex car)))
          (pattern (dots article nnp dots))
          (nnp ((cat nnp))))
```

6) The top-level constituent **(cat np)** is unified. The **cset** contains: **article** and **nnp**. These two constituents are unified in turn. We will show part of the unification process for **nnp**.

When the **nnp** constituent is unified, the input is:

```
(nnp ((cat nnp)))
```

Note that this input does not really contain all the elements it should contain: had the input been completely specified, the input would be:

```
(nnp ((adj ((lex big)))
      (nnp ((adj ((lex red)))
            (nnp ((adj ((lex old)))
                  (nnp ((n ((lex car)))))))))))
```

However, because we allowed for the input to be loosely specified, we do not have such a structure. As we must be able to unify the structure we have, we need to have access to the constituents that were allowed to be unattached and are thus not part of the current input. This is done through *loose*.

In this case, while unifying **(nnp ((cat nnp)))** with the grammar, the program searches the input for an **adj**. In the completely specified input, it would find the adjective inside the input. Here, however, as the adjective was allowed to be "unattached", it is not found at this level. It is stored into *loose*.

How Loose is Handled

We now modify the explanations for the functions used by the unifier:

1. First, one argument is added: *loose*. Most functions are called with this argument in addition to the arguments we already mentioned.

2. After unifying a constituent, all attributes of the input that have not been used are added into *loose*.

3. **Cntrl-av-h-ls**: When looking for an attribute value pair in the input, **cntrl-av-h-ls** looks into *loose* if the pair is not found in the input itself.

In the example, when (cat nnp) is unified, we first have an alternative in the grammar. The first alternative is tried:

((adj none) (pattern (n pound)) (n ((cat noun) (lex any))))

This alternative fails because, when looking for a pair with attribute *adj*, **cntrl-av-h-ls**, failing to find such a pair in the input, looks into *loose* and find (adj ((lex big))). The value ((lex big)) fails to unify with *none*.

The next alternative is taken:

((adj any) (adj ((cat adj) (lex any))) ...)

Again, when unifying the input (nnp ((cat nnp))) with (adj any), **cntrl-av-h-ls** fails to find a pair with the corresponding attribute in the input and looks into *Loose* for it.

(adj ((lex big))) is found in *loose*, taken from it and added to the input which becomes:

(nnp ((cat nnp) (adj ((lex big))))).

The dajective is now attached to the appropriate constituent. The unification proceeds.

We see from this example that, with *loose*, we are able to simplify the input by having unattached constituents.

3. **Ucon-proc**: **Ucon-proc** calls **uavs-ls** on the constituents indicated in the cset. Before unifying these constituents, *loose* is updated so that unattached constituents at this level can be accessed at lower levels (just like, in the example above, the adjectives were available at the NNP level, which is a level lower than the NP level). Thus, before unifying the constituents, **ucon-proc** calls a function to update *loose*.

4. Finally, when an attribute value pair is taken from *loose* during unification, *loose* is updated to reflect this change: the pair is deleted from *loose*.

6. Tracing facilities

Tracing facilities have been added to the unifier for an easier tracing of the unification process. This is useful both for a person building the grammar[14] and a person changing the unifier.

Because of the recursiveness of the unification, tracing can produce a lot of output. The following tracing variables are an attempt to curb that effect.

- *top-level-tracing*: This variable traces the different steps in the function **unify**. It shows the result of the unification of the highest constituent before **uavs-ls** is called again on the constituents contained in the *cset*. It also shows the output of the unification process, before the linerizer is called. This is useful to see the deep structure produced by the unifier, and to separate the two stages, unification and linearization.

- *inside-tracing*: This variable traces each call to **uavs-ls**, and a few other functions. It thus traces in a fairly complete manner the unification process. It generates a lot of output however. Since this variable traces the whole unification, it is useful mainly to get an understanding of the process. More selective tracing is preferable when one wants insight into a particular problem.

- *tracing-get-val-if-path*: This variable traces the calls to the function get-val-if-path, the function that finds a value in a higher FD when the grammar value contains an *up-arrow*. This is useful when one wants to trace only the value found for a path with an *up-arrow*.

- *tracing-ucon*: This variable traces the unification of the constituents contained in the *cset*. It shows the *cset*, which constituent is being unified, and how *loose* is updated before unifying a constituent.

- *select-grammar-part*: This variable allows the user to trace only part of the grammar, for example, (cat np). The user can set this variable to the part(s) that is (are) to be traced: (SETQ SELECT-GRAMMAR-PART '((CAT NP) (CAT ARTICLE))) for example. *Select-grammar-part* sets *inside-tracing* to true during the appropriate sub-grammar to trace the unification process at that time.

- *select-constituent*: This variable is set to trace a constituent (as specified by the *cset*. As an example, (SELECT-CONSTITUENT '(PROT)) traces the unification process when *prot* is unified. As for select-grammar-part, this variable sets *inside-tracing* to true when necessary.

---

[14] A person constructing the grammar can find out where the grammar fails by looking at the trace. Also, once the grammar is developed, tracing the unification may result in finding inefficiency in the grammar.

- *trace*: If we only want to know where the unification process fails, it is useful to trace the function **uavs-ls** ***only with*** the input and the grammar. This is done with *trace*.

- *trace-loose*: *Trace-loose* is used if, in addition to the input and the grammar, the user want to trace the variable *loose* at each callto uavs-ls. *Trace-loose* cannot be used alone: The variable *trace* needs to be set also for any tracing to occur when *trace-loose* is set.

7. Problems encountered during summer 85

We found three kinds of problems. I will explain here what these problems were, why we think they occurred, and how we solved them[15].

Wrong agreement of the main verb

Problem:

In the active (passive) voice, the verb should take the number of the protagonist (goal). However, when the protagonist (goal) is a subordinate (as opposed to a noun), whenever there is a plural noun in the subordinate, the verb also ends up in plural.

Examples:

Active voice; the goal of the subordinate is plural:

```
(THAT A PERSON SPEAKS WORDS CAUSE A WAVE)
```

Passive voice; one noun in the subordinate is plural:

```
(THAT MOLECULES MOVE ARE CAUSED)
(THAT A MOLECULE IS MOVED BY WAVES ARE CAUSED)
(THAT A WAVE MOVES MOLECULES ARE CAUSED BY A PERSON)
(THAT WAVES MOVE MOLECULE ARE CAUSED BY PERSONS)
```

Hypothesized Cause:

We think this problem occurs because, when searching for a value into a functional description, the search is carried as deep as necessary to find the number (or other attributes in general). In case of a subordinate, we have the following:

```
(prot ((embed ((relpro == that)
               (verb ((v == compress)))
               (prot ((n == diaphragm) (article == def)))
               (goal ((n == granule)
                      (number plur)
                      (article == def)))))))
```

"Plur" (plural) is returned, because ***number*** was found in the goal of the subordinate. (The same would have happened, had the protagonist been plural.)

Change made to solve this problem:

---

[15]These problems have been documented in TjoeLiong's DOC files and Cecile's TRY files.

This search was done by a call to the function **assoc-av**. I changed this function so that it only looks at the top elements of the list.

More number problems

The problem

Unification fails in some case, when both protagonist and goal are subordinates and their respective subjects do not have the same number. In other cases, the verb is in the passive voice, when the voice active should have been defaulted. This occurred because the first alternative in the grammar, voice active, failed because of a number problem.

Examples:

The following intended sentences failed:

```
A WAVE CAUSES THAT MOLECULES MOVE
THAT A PERSON SPEAKS CAUSES THAT MOLECULES MOVE
```

The following input resulted in passive voice,
while active should have been the default:

```
(SETQ Q6
      '((CAT S) (VERB ((V == CAUSE)))
        (PROT ((N == PERSON) (ARTICLE == INDEF)))
        (GOAL ((SUBORD ((RELPRO == THAT)
                        (S
                         ((VERB ((V == MOVE)))
                          (GOAL ((N == MOLECULE)
                                 (ARTICLE == INDEF)))
                          (PROT
                           ((N == WAVE) (NUMBER PLUR)
                            (ARTICLE == INDEF)))))))))))
```

```
(A PERSON CAUSES THAT A MOLECULE IS MOVED BY WAVES)
```

Hypothesized Cause:

When the grammar indicates that a value is to be found in the input (in case of the *up-arrow*), the variable GVAL (grammar value) gets set to the value returned by **get-val-if-path**, the function that looks for the value.

Then, **cntrl-av-h-ls** looks into the input for the value. The *up-arrow* indicated that the value should be found not in the current FD, but in the higher FD. So, of course, no value is found in the input. **Cntrl-av-h-ls** then looks into *loose* for the value. If, because of other unattached constituents, it finds a number value in *loose*, especially *number none*, that value gets picked up, and the two values are unified. If **get-val-if-path** returned "plur" for the grammar value, thr program tries to unify *plur* and *none*, and unification fails.

Change made to solve this problem:

To solve this problem, as the value given to the grammar (*GVAL*) was taken in fact from the input, it is also assigned to the input value *(IVAL)*. [So both ***gval*** and ***ival*** would get ***plur***.] Both functions **Av-h-ls** and **cntrl-h-ls** were changed.

Attributes being borrowed from other constituents

The problem

Consider the following input: two coordinated clauses. One of the clause has a protagonist and a goal. The other one has only a goal. Unification resulted in assigning the protagonist from the first clause to both sentences:

Example[16]:

```
(SETQ N1
      '((CAT S) (CONJ == SINCE) (ORDER PRE)
                (S1 ((VERB ((V == COMPRESS) (VOICE PASSIVE)))
                     (PROT ((N == GRANULE) (NUMBER PLUR)
                            (ARTICLE == INDEF)))
                     (GOAL ((N == TRANSMITTER) (NUMBER PLUR)
                            (ARTICLE == INDEF)))))
                (S ((VERB ((V == MOVE)))
                    (GOAL ((N == WAVE) (NUMBER PLUR)
                                       (ARTICLE == INDEF)))))))
```

Output:
(SINCE TRANSMITTERS ARE COMPRESSED BY GRANULES GRANULES MOVE WAVES)

The second clause ''borrowed'' *granules* from the first one.

Intended output:
SINCE TRANSMITTERS ARE COMPRESSED BY GRANULES, WAVES ARE MOVED

Hypothesized Cause:

After processing a functional description, *loose* is updated to contain the unattached elements before the constituents indicated by the *cset* are unified.

```
The top-level (CAT S) level has been unified.
Before S1 and S are unified,  Loose gets the
unattached constituents from this level.
Loose has value L1.
```

The only other time *loose* gets updated is when items are taken from it. Now, while *S1* is unified, more elements are added to *loose*. These remains in *loose* when the next clause is unified. This is a mistake, as, even with unattached constituents, there are no reasons why the unification of the second clause should depend on attribute value pairs that were added to *loose* when the first clause (prot), its sibling node, was processed[17].

---

[16]The grammar corresponding to this example was developed by TjoeLiong Kwee a long time ago and is not used anymore. The problem however, is independent of this grammar

[17]A similar problem occurs in a sentence with two subordinates. A protagonist (goal) is sometimes borrowed from one of the subordinates and added to the other

Change made to solve this problem:

Ucon-proc was changed so that *loose* recovers its original value before processing the next constituent (a sibling node). It is still updated when elements are taken from it.

8. Changes to the Unifier

This section describes the changes I have made to the unifier.

1. Changes to solve problems: These were mainly described in the previous section.

2. Changes for efficiency purposes: The main changes were described when the function **alt-h-ls** was explained. They concern the lexicon and the handling of alternatives.

3. Other minor changes were made:

   - A new function was added to check whether **get-val-lf-path** should be called. This function essentially does the test **get-val-lf-path** does at the beginning. Since, most of the time, **get-val-lf-path** does not need to be called, it was simpler to include the test before calling the function. (It was also very useful in tracing the unifier to delete needless calls to the function **get-val-lf-path**.

   - Similar changes were done in other functions.

4. I added comments to the code.

5. Finally, I eliminated the existing tracing variables and added others, which I think are more useful (see the section on tracing facilities).

9. Remaining problems

A list of remaining problems follows:

1) It is very hard to control when unattached constituents should be taken from *loose*. Sometimes, a constituent is taken from *loose* when it should not. Most of the time it does not cause any problems in the result (although it is theoritically wrong):

Example:

Consider a sentence in which the protagonist is a subordinate clause. After unifying the top-level constituent (CAT S), *loose* will contain elements such as: (CONJ NONE), (TAIL NONE), etc... (pairs that were added to the input). The constituents are then unified: prot is unified. It is also of (CAT S). When unifying this constitent with the grammar, at some point the pair (CONJ NONE) from the grammar is unified. The program does not find a pair with the same attribute in the input and thus looks into *loose* for it. It will find (CONJ NONE) and return this pair. Note that this does not produce wrong results but is wrong as (CONJ NONE) applies to the top-level sentence.

However, it does create problems sometimes, in which case we are forced to specify part of input (to force the attribute to be in the input). This is contrary to the motivation for *loose*. A more important problem is that we don't know ahead of time when it will be necessary to specify the input more completely. This brings the question whether it is really possible to have a very unspecified input[18].

2) Features, such as NUMBER. Features do not seem to be handled in a defined way. This creates problems sometimes (apparently especially in the linearizer). For example, in some cases, when there is a noun which is plural in the sentence, the indefinite article disappears, no matter where in the sentence this article is (i.e. it is not always the article attached to the plural noun). See Appendix V for an example.

3) Gap: Gapping is still a problem. TjoeLiong Kwee mentions it also in his report. The main problem of the *gap* is to refer to its antecedent, when the gap is in the position of a relativized constituent , so that number agreement can take place.

4) Low level details: Some of the list manipulations done is messy and counts on side effects (which are not obvious). It would be desirable to "clean" the code.

5) We are not taking advantages of the potential of the functional grammar: we yet have to add focus and other "attributes" in the grammar.

---

[18]TjoeLiong mentions this problem. I could not find the examples we had encountered before.

10. Progress on the grammar, summer 85

This section reports on the changes made to the grammar by TjoeLiong Kwee during summer 85. The report indicates which parts of the grammar have been tested, expanded, or changed. The report includes the test inputs used, corresponding outputs, the changes made to the linearizer, and the final grammar [19]

[19] The final grammar is the complete grammar taken from [McKeown 82], with the changes reported here included. As this is the complete grammar from [McKeown 82], note that parts of it remains to be tested.

Owner	KWEE
Name	PS:<KWEE>TL-REPORT-03...
[Date]	22-Aug-85 4:15:21PM
Jobheader	
PageCollation	
FormLength	66
Language	Printer

FILE <KWEE>TL-REPORT-00.LSP

8 Aug 1985 1740-EDT
18 Aug 1985 1355-EDT

TJOELIONG KWEE : DOCUMENTATION on <KWEE>TL-FULLGRAM-8.LSP

REPORT on proposed modifications in <MCKEOWN.VAX>GRAMMAR.LSP
carried out during the periods april-may and july-august 1985

Up till now the following parts of the original grammar have been
studied (where mentioned : modified and/or added) , and tested:

- CAT S : VERB-VOICE : ACTIVE , PASSIVE : PROT , GOAL (modified , added)
- CAT VERB-GROUP : VOICE : ACTIVE , PASSIVE (modified)
- CAT S : CONJ ; CAT SLIST
- CAT NP : CONJ ; CAT NPLIST
- CAT S : SUBORD (added)
- CAT NP : EMBED (added)

Moreover , some changes have been made in <MCKEOWN.VAX>LINEAR.L ,
and at the same time <MCKEOWN.VAX>UCON.L has been modified ,
first by Michal Blumenstyk and then again by Cecile Paris .

The selection of the grammar fragments has mainly been motivated
by the need for richer output structures in general , and
in particular when the grammar was used as sentence-generating tool
in research done by Cecile Paris (especially for default passive
when no protagonist is given at input , and for subordinate clauses) .
As such , the work on the grammar is not yet finished ;
some desiderata and other problems will be mentioned
at the end of this report .

OVERALL STRUCTURE OF THE GRAMMAR

The Grammar as a whole is one large Functional Description (see Kay 1979)
in which eventually all possible structures of an English sentence
are to be found . An input structure is unified with the grammar ,
and if this input is compatible with some part of the grammar ,
an output follows . Otherwise the result will be empty (NIL) .
At the topmost level , the grammar offers an alternative between a number
of categories ; the input should have a category as well .
In this report we will be concerned with the categories S and NP .

CONTENTS OF THIS REPORT :

1) VERB-VOICE ACTIVE AND PASSIVE

At the highest level immediately under CAT S an alternative is offered between simple main clause and conjoined main clauses : here we treat only the first part .
Of the three choices to be made there in the original version - for verb-voice , sentence-adverbial , and sublist - we treat only the first one : that for a verb-voice .

1.1) CAT S : VERB-VOICE

In the original version any input sentence without explicit mention of VERB-VOICE is incompatible with the last two of the three options given :

```
((verb ((voice any))) (verb ((voice passive))) ... ) ,
((verb ((voice any))) (verb ((voice there-insertion))) ... ) ,
```

but compatible - and therefore unified - with the first option :

```
((verb ((voice active))) ... ) ;
```

this however requires the input sentence to have an explicit PROT - as already has been mentioned in McKeown 1982 p.216 .

In order to accommodate the case of an input sentence without PROT and without explicit mention of VERB-VOICE , I propose some modifications in the grammar .
At the same time I propose other minor changes that - as a side result of this investigation - turn out to be feasible.

Essentially , the description I propose (with the help of Cecile Paris who found its most concise and efficient formulation) has this skeleton :

```
(alt
 (
% ------------------------------------------------
  ((verb ((voice active)))
   (prot any) (prot ((cat np)))
   (verb ((cat verb-group) (number (^ prot number))))
   (pattern (prot dots verb dots pound))
   ( . . . ))
% ------------------------------------------------
  ((alt
    (((verb ((voice passive)))
      (prot none))
     ((verb ((voice any) (voice passive)))
      (prot any) (prot ((cat np)))
      (by-obj ((cat pp) (prep ((lex by))) (np (^ prot))))
      (pattern (dots verb by-obj dots pound)))
    ))
   (verb ((cat verb-group)))
   ( . . . ))
% ------------------------------------------------
  ((verb ((voice any) (voice there-insertion)))
   ( . . . ))
% ------------------------------------------------
  ))
```

This part of the grammar has been tested for input containing PROT and/or GOAL and/or PASSIVE . The grammar has been completed with the part from the original version as regards BENEF - but it has not yet been tested on input in which a BENEF constituent is given ; likewise , the parts concerning VERB-VOICE THERE-INSERTION have been added but not yet tested on input .

1.2) CAT VERB-GROUP

Changes within the VERB-VOICE alternative at the CAT S level , however , although combined with corresponding changes at the CAT VERB-GROUP level , are not sufficient to obtain "passive by default" in all and only the intended cases ; the main source for errors seems to reside in the description at this latter level ; in the original version there are two problems : first , an input without specified VOICE is compatible - and as a consequence , unified - with (voice active) : a point that has to be modified in the first place , since global unification will fail if there is no PROT ; second , (voice passive) splits up into (prot none) and (prot any) , where the last one is an "empty choice" , since there is no PROT at the level of CAT VERB-GROUP anyway .
Various attempts to maintain the (prot any) there as a "feature" , by adding the same "feature" (prot any) at an appropriate place in the corresponding VERB constituent description somewhere under CAT S , failed .
After having tried this in vain , however , I found that the grammar could be freed of some more duplication between on one hand CAT VERB-GROUP and on the other hand VERB constituents within the PROT/GOAL choices under CAT S .
In particular , I have been able to remove from the CAT VERB-GROUP level :
- the option (voice none) , since the voice must already have been added before , when unifying the input at the CAT S level ;
- the (pattern (... by-obj ...)) , by moving it to a more appropriate and "natural" place under the CAT S level .

```
((cat verb-group)
 (alt
  (((voice active) (pattern (v dots))
    (v ((cat verb) (lex any))) )
   ((voice passive) (pattern (v1 v dots))
    (v1 ((cat verb) (lex be)))
    (v ((cat verb) (lex any) (tense pastp))) )
   ((voice there-insertion) (pattern (v dots))
    (v ((lex any) (cat gap) (gap +))))))
 ( . . . ))
```

The grammar at this point has been tested on the six input structures given in file TL-TEST-1.LSP .

Note . During earlier experiments with the original version I met with the following strange phenomenon : an input with structure

```
((cat s) (verb ((v === vvv) (voice PASSIVE)))
         (prot ((n === ppp) (article === INDEF)))
         (goal ((n === ggg) (number PLUR) (article === ...))) )
```

resulted in an output of the form : [THE] GGG ARE VVV-ED BY PPP
that is , the singular indefinite article A[N] would not appear .
This only happened with SINGULAR INDEFINITE protagonist/by-object
combined with PLURAL goal/subject ,
and not with (singular or plural) definite protagonist/by-object
or with singular goal/subject .
The same input , however , unified with the grammar in which my proposed modifications are incorporated , results in correct output - that is , the singular indefinite article A[N] will now show up correctly .
This can only be ascribed to (pattern (... by-obj ...)) being moved from under CAT VERB-GROUP in the original version - where , as it turns out , it will never be reached because of the (prot any) that never applies - to under CAT S in the proposed modified version , right where we find (verb ((voice passive)) (prot any) (by-obj ...)
- which indeed seems to be a more "natural" place for it .
It remains an unsolved riddle , though , how this can influence the [dis]appearance of an indefinite article in the protagonist noun-phrase .
Note : just the same transposition in the original grammar led only to pushdown overflow at repeated runs .

2) SUBORDINATE CLAUSES

As for the three different types of subordinate clauses (see section 2.0) ,
I have undertaken steps to generate :
- subordinate clauses in Adverbial-position ,
- subordinate clauses in NP-position ;
the generation of Relative Clauses has yet remained a desideratum
(see section 2.3 for results as of date) .

The original version of the grammar provides for (very simple) Relative Clauses
in the following way , within CAT NNP :

```
((cat nnp)
 (alt
  ((( . . . ))
   (( . . . ))
   ((tail any)
    (tail ((cat s-bar)))
    (nnp ((cat nnp)))
    (pattern (nnp tail))
    )
   )))
```

where CAT S-BAR is defined as :

```
((cat s-bar)
 (s ((cat s)))
 (pattern (dots s))
 (alt
  (((relpro none))
   ((relpro any)
    (relpro ((cat relpro) (lex any)))
    (pattern (relpro dots))
    ))))
```

and the lexical category RELPRO contains the relative pronouns
THAT , WHO , WHICH , etc.

For subordinate clauses in general , so it seems , we can use just the same
CAT S-BAR , provided we expand the lexical category RELPRO - preferredly
renamed as COMP - to include henceforth the appropriate complementizers
such as , for NP-subordinates : THAT , and for Adv-subordinates :
AFTER , BEFORE , BECAUSE , SINCE , UNTIL , WHEN , WHILE , etc.

Subordinate Clauses in Adv-position are described within CAT S
(section 2.1) , those in NP-position within CAT NP (section 2.2) ;
Relative Clauses have not yet been studied extensively :
some problems that have come up initially are mentioned in section 2.3 .

A very short introduction to Subordinate Clauses
for linguistic naive computer scientists
is added as a supplementary section 2.0 .

2.0) SHORT INTRODUCTION TO SUBORDINATE CLAUSES FOR THE LINGUISTIC NAIVE

" how it is in language "

1) the basic unit is the (Simple) Sentence .
2) a Sentence consists (basically) of a Verb and
   some Arguments , like Subject , Direct Object , etc.
   example : A WAVE (Subject) MOVES (Verb) THE MOLECULES (Direct Object) .
3) a Simple Sentence has exactly one (Finite) Verb .
4) if there is more than one (Finite) Verb, then we have
   - either a Compound Sentence , if all components are on the same level :
     these are Coordinated Clauses , connected to each other by a Connective ;
     example: A PERSON SPEAKS (Cl. 1) AND (Connective) THE WAVES MOVE (Cl. 2) .
   - or a Complex Sentence , if one is at top level (the Main Clause)
     and the other at a lower level (the Subordinate Clause) .
5) a Subordinate Clause usually is introduced by a Complementizer ,
   which signals that what follows is not an independent clause -
   but beware , sometimes the Complementizer can be "deleted" !
6) there are three different kinds of Subordinate Clauses ,
   and which kind it is , depends on its position in the higher Clause :
   - either in Noun-Phrase position ; example :
     THE SOUND CAUSES (Verb) A WAVE (Direct Object , Noun Phrase) , versus
     THE SOUND CAUSES (Verb) THAT MOLECULES MOVE (Subordinate in NP-position) ;
   - or in Adverbial-Phrase position ; example :
     THE WAVE MOVED (Verb) YESTERDAY (Adverbial-Phrase) , versus
     THE WAVE MOVED (Verb) BECAUSE A PERSON SPEAKS (Subordinate in Adv.pos.) ;
   - or as a Modifier of a Noun (inside a Noun-Phrase) ; example:
     THE LARGE (Adjective , modifying the Noun) MOLECULES , versus
     THE MOLECULES (Noun) THAT MOVE (= Relative Clause modifying the Noun) .

" how it is in the programs <MCKEOWN.VAX>GRAMMAR.LSP & <KWEE>TL-FULLGRAM-8.LSP "

7) any (simple , compound or complex) sentence at top-level is of CAT S .
8) coordination of clauses is described by way of CONJ , which connects
   an SLIST with an S ; the SLIST itself is either just an S , or
   (recursively) an SLIST plus an S , but without CONJ .
9) any Subordinate Clause is essentially of CAT S-BAR .
10) an S-BAR is , essentially , an S optionally preceded by a Complementizer ;
    the original grammar has only a category of relative pronouns RELPRO
    (example : THAT as a relative pronoun , WHO , WHICH , etc.) ;
    we will need more Complementizers , like AFTER , BEFORE , BECAUSE , SINCE ,
    UNTIL , WHEN , WHILE , etc. so I would propose to rename RELPRO into COMP .
11) in order to be able to distinguish the different roles of Subordinates , or
    rather : the different positions , these have to be differentiated as such :
12) a Subordinate Clause that is intended to occupy an NP-position ,
    is of CAT NP , but to distinguish it from a "normal" Noun Phrase ,
    I "subcategorized" it as EMBED (maybe NPSUBORD would have been clearer?) ;
    example : A WAVE CAUSES THAT MOLECULES MOVE :

```
(setq x '((cat s)
          (verb ((v === cause)))
          (prot ((n === wave)(article === indef)))
          (goal ((embed ((relpro === that)
                         (s ((verb ((v === move)))
                             (prot ((n === molecule)(number plur)))))))))))
```

13) a Subordinate Clause that is intended to occupy an Adverbial position maybe
    should have been put in CAT ADV , and distinguished in a similar way , but
    I put it in CAT S and "subcategorized" it as SUBORD (rather ADVSUBORD ?) ;
    example : MOLECULES MOVE, BECAUSE A PERSON SPEAKS :

```
(setq x '((cat s)
          (s ((verb ((v === move)))
              (prot ((n === molecule)(article === indef)(number plur)))))
          (subord ((relpro === because)
                   (s ((verb ((v === speak)))
                       (prot ((n === person)(article === indef)))))))))
```

14) a Subordinate Clause that is intended to modify a Noun is part of a
    Noun-Phrase ; the original grammar describes it as TAIL within CAT NNP .

2.1) SUBORDINATE CLAUSE IN ADV-POSITION : CAT S , SUBORD

The fact that a Subordinate Clause in the role of an Adverb can appear
either in sentence-initial or sentence-final position -
that is , both Subord + Main Clause , and Main Clause + Subord
are possible ; example : BECAUSE A PERSON SPEAKS, WAVES MOVE
and : WAVES MOVE, BECAUSE A PERSON SPEAKS -
and my desire to reflect this in the grammar , initially led to problems
with (pattern (. . .)) .
In one version I tried to treat SUBORD as a sister constituent of VERB ,
PROT , and GOAL , having as one of the possible orders :
(pattern (subord dots)) , the other one being (pattern (dots subord pound)) ;
both , however , conflict with (pattern (prot dots verb dots goal pound)) ;
after several experiments I found this solution :
put another alternative (alt (((subord none) (-)) ((subord any) (-))))
in CAT S , within the CONJ NONE part ; the VERB-VOICE alternative should
completely be under ((subord none) (-)) :

```
((cat s)
 (alt
  (((conj none)
    (alt
     (((subord none)
       (alt - - - VERB VOICES - - - ))
      ((subord any)
       (subord ((cat s-bar)))
       (s ((cat s)))
       (alt
        (
         % ------------------------------------------------
         % SUBORD IN SENTENCE-FINAL POSITION - BY DEFAULT
         ((order post) (pattern (s subord))
          (s ((punctuation ((after ","))))))
         % ------------------------------------------------
         % SUBORD IN SENTENCE-INITIAL POSITION
         ((order any) (order pre) (pattern (subord s))
          (subord ((punctuation ((after ","))))))
         % ------------------------------------------------
         )))
      )))
   ((conj any - - - etc. ))
  )))
```

This gives satisfying results but has , at first sight , the drawback
of requiring rather specified input , since that should look like :

```
(setq x '((cat s) (order pre)
          (s ((verb ((v === compress)))
              (prot ((n === transmitter) (article === def)))))
          (subord ((relpro === because)
                   (s ((verb ((v === move)))
                       (prot ((n === wave) (article === def)))))))))
% output: (BECAUSE THE WAVE MOVES!, THE TRANSMITTER COMPRESSES!.)
% without mention of order , or with (order post) , the output is :
% (THE TRANSMITTER COMPRESSES!, BECAUSE THE WAVE MOVES!.)
```

Note : the actual version of the unifier , however , is such that it allows
us to simplify the input by omitting the S at main level and inside SUBORD .

This part of the grammar has been tested on all 4 x 4 possible
combinations of PROT , GOAL , PROT+GOAL , PROT+GOAL+PASSIVE
for each of S and SUBORD , both without ORDER and with ORDER PRE .
For the 16 input combinations with an even distribution of both orders see
file TL-TEST-2.LSP ; for input with simplification for S , file TL-TEST-3.LSP .

2.2) SUBORDINATE CLAUSE IN NP-POSITION : CAT NP , EMBED

For NP-Subordinate Clauses (embedded sentences) ,
we need a slightly richer description of CAT NP ;
the CAT NP of the original version has been modified at the highest level :

```
((cat np)
 (alt
  (((embed none)
    ( . . . text taken from original grammar . . . ))
   ((embed any)
    (embed ((cat s-bar))) (pattern (dots embed dots)))
  )))
```

Input has to look like :

```
(setq x '((cat s)
          (verb ((v === ... )))
          (prot ((embed ((relpro === that)
                         (s ( ... ) )))))
          (goal ((embed ((relpro === that)
                         (s ( ... ) ))))) ))
```

Note : the unifier , however , is such that it is not necessary to
specify at input the S inside the EMBED ; some sample input to test this :

```
% ----------------------------------------------------------------------
(setq x '((cat s)
          (verb ((v === cause)))
          (prot ((embed ((relpro === that)
                         (verb ((v === speak)))
                         (prot ((n === person)(article === indef)))))))
          (goal ((embed ((relpro === that)
                         (verb ((v === hit)))
                         (prot ((n === wave)(number plur)))
                         (goal ((n === diaphragm)(article === def)))))))))
% intended output : that a person speaks causes that waves hit the diaphragm.
% ----------------------------------------------------------------------
(setq x '((cat s)
          (verb ((v === cause)))
          (prot ((embed ((relpro === that)
                         (verb ((v === speak)
                                (pp ((prep === into)
                                     (np ((n === transmitter)
                                          (article === def)
                                          (pp ((prep === of)
                                               (n === telephone)
                                               (article === indef)))))))))
                         (prot ((n === person)(article === indef)))))))
          (goal ((embed ((relpro === that)
                         (verb ((v === hit)))
                         (prot ((n === wave)(adj === sound)
                                (article === def)(number plur)))
                         (goal ((n === diaphragm)(article === def)
                                (pp ((prep === of)
                                     (n === transmitter)
                                     (article === def)))))))))))
% intended output : that a person speaks into the transmitter of a telephone
% causes that the sound waves hit the diaphragm of the transmitter.
% ----------------------------------------------------------------------
```

For the first input , as well as the variant with a non-finite form
(infinitival) of the subordinate verb , see file TL-TEST-4.LSP .

2.3) RELATIVE CLAUSE : CAT NNP , TAIL

I understood that CAT GAP primarily served to "produce" the "deleted" relativized constituent in a Relative Clause (see the examples below) .

The original grammar provided only a GAP alternative for PROT :

```
(prot ((alt (((cat np) (gap none))
             ((cat gap) (gap any) (gap +))))))
```

I added the same alternative for GOAL (not yet for BENEF) , and as a first try I tested these inputs :

- voice active , prot gap +
  intended output : A WAVE THAT (gap) COMPRESSES A DIAPHRAGM

- voice active , goal gap +
  intended output : A DIAPHRAGM THAT A WAVE COMPRESSES (gap)

- voice passive , goal gap +
  intended output : A DIAPHRAGM THAT (gap) IS COMPRESSED BY A WAVE

- voice passive , prot gap +
  intended output : A WAVE THAT A DIAPHRAGM IS COMPRESSED BY (gap)

Results and first conclusions :

- the fourth input form failed ,
  most probably because of (by-obj ((np (^ prot)))) with (prot ((gap +))) :
  BY-OBJ is of CAT NP , but PROT is of CAT GAP ;
  at first sight , a possible solution would be to include a constituent n-gap under cat np , but that could lead to undesired results , like for instance THE MOLECULES, THE GRANULES AND (gap) ; in general , the grammar in that case would allow to have a GAP in any single NP position , so one would depend on correctness of the input ;
  I suggest rather a new CAT N-BAR which splits up into an N-GAP and an NP , and the grammar then having (cat n-bar) instead of (cat np) in appropriate places , as prot , goal , and inside the pp (in order to account for the "gapped" by-object) ; but both these suggestions need further investigation .

- the first three input forms resulted in output with
  the verb always in third person singular ,
  because it takes its NUMBER from (prot/goal ((gap +))) ,
  which has no number , and so by default is supposed to be singular ;
  solving the problem of accessing the antecedent of a Relative Clause belongs to the desiderata .

For complete and commented input and output see files TL-TEST-5.LSP/OUT .

2.4) NON-FINITE FORM OF VERB : ASPECT

As a side effect of investigating NP-subordinate clauses ,
I managed to generate forms with non-finite verbs as well :
this is a matter of some minor additions in the Linearizer ,
for the listing of which the reader is referred to the Appendix .
The result is that , when , at input , one gives VERB the additional
"feature" ASPECT - which must have one of the two values INF or ING - ,
at output the S-BAR will correspondingly have either
the infinitive or the ing-form of the verb ;
the infinitival form is useful for embedded sentences ,
the ing-form serving optionally for subordinates in adverbial position .

sample input (note : the unifier , however , is such that it is not
necessary to specify at input the S inside EMBED/SUBORD or at Main Clause) :

```
(setq x '((cat s)
          (verb ((v === cause)))
          (prot ((embed ((relpro === that)
                         (s ((verb ((v === speak)))
                             (prot ((n === person)(article === indef)))
                             ))))))
          (goal ((embed ((s ((verb ((v === hit)(aspect inf)))
                             (prot ((n === wave)(number plur)))
                             (goal ((n === diaphragm)(article === def)
                                    ))))))))))
```

output :

(THAT A PERSON SPEAKS CAUSES WAVES TO HIT THE DIAPHRAGM!.)

```
(setq x '((cat s)(order pre)
          (s ((verb ((v === compress)))
              (prot ((n === diaphragm)(article === def)))
              (goal ((n === granule)(article === def)(number plur)))))
          (subord ((s ((verb ((v === move)(aspect ing)))
                       (prot ((n === wave)(article === def)))
                       (goal ((n === molecule)
                              (number plur)(article === def)))))))))
```

output :

(THE WAVES MOVE-ING THE MOLECULES!, THE DIAPHRAGM COMPRESSES THE GRANULES!.)

For test input for non-finite verb forms , as well as input for finite forms ,
see file TL-TEST-4.LSP .

Note . ASPECT is not mentioned in the proposed version of the grammar ,
but when it is given at input , the proposed additions in the Linearizer will
produce the correct output . Kathy McKeown and Cecile Paris both suggested
independently from each other that the grammar should infer ASPECT
in some way or another , for instance from the absence of RELPRO .
It's not that easy , however , because of the following observations :
- RELPRO can be omitted as well from an S-BAR intended as a Relative Clause ,
  example : THE GRANULES THE DIAPHRAGM COMPRESSES CONDUCT THE ELECTRIC CURRENT ;
- RELPRO can be omitted as well from an EMBED under verbs like SAY , BELIEVE ,
  etc. ; example : THEY BELIEVE IT CAN BE DONE IN SOME WAY OR ANOTHER ;
moreover , I prefer to leave the choice between finite and non-finite verb form
a free option whenever possible .

3) DESIDERATA : WHAT REMAINS TO BE DONE

Here follows a list of things that have not yet been completely studied , or not studied at all , but clarification of which should be very desirable :

3.1)

- Relative Clauses , GAP , reference to Antecedent Noun .
- Noun Phrases in general , with all related constituents like ARTICLE , POSSESSIVE , ADJ , PP , TAIL , especially in nested constructions ; for instance , the original grammar describes POSSESSIVE as CAT NNP , and gives a (feature possessive) to the N , which makes the Linearizer put the "'s" (apostrophe-s) next to the Noun ; now two questions arise :
  i) since NNP can be [[N PP] TAIL] , how should its possessive look like ? the original version gives "[[N-'s PP] TAIL]" where "[[N PP] TAIL]-'s" is preferable at least in spoken English at least to some native speakers ;
  ii) POSSESSIVE is described in the grammar as part of CAT NP , not part of CAT NNP ; yet we would like to have an NNP with a possessive , that is , [NNP-'s NNP]-'s NNP ; for example in English it is quite possible to have a construction like :
  THE MODERN TELEPHONE'S SMALL TRANSMITTER'S METAL DIAPHRAGM .

3.2)

- is the "feature" NUMBER connected with Noun Phrases (NP , NNP) or Nouns ?
- how is NUMBER related to Verb Agreement ? (for example in Relative Clauses , and with Conjoined Subjects)

Note : possibly these two problems are more connected with the Linearizer , since it is mostly a matter of Morphology .

3.3)

The grammar fragments that up till now have not been considered yet :

- BENEF .
- THERE-INSERTION .
- SENTENCE-ADVERBIAL within CAT S .
- ADV and PP within CAT VERB-GROUP .

3.4)

Most importantly , and essentially related to the advantages proper of Functional Unification Grammar as a powerful notation (Kay 1979) :

- the treatment of syntactic and pragmatic functions like Subject , Direct Object , Indirect Object , Focus , Topic , etc.

3.5)

- Matters morphological in general : verbs , plural , possessive .
- The Linearizer in general , PATTERN .

Owner	KWEE
Name	PS:<KWEE>TL-TEST-1.LSP.3
[Date]	18-Aug-85 2:03:50PM
Jobheader	
PageCollation	
FormLength	66
Language	Printer

```
% TL-TEST-1.LSP : for default passive etc.
% ------------------------------------------------------------------------
% 1 : VOICE ACTIVE, PROT NONE : (nil)
(setq x
  '((cat s)
    (verb ((v == compress) (voice active)))
    (goal ((n == wave) (article == def) (number plur))) ))
(pp x)
(unify x grammar)
% ------------------------------------------------------------------------
% 2 : VOICE ACTIVE, PROT ANY : (a diaphragm compresses the waves)
(setq x
  '((cat s)
    (verb ((v == compress) (voice active)))
    (prot ((n == diaphragm) (article == indef)))
    (goal ((n == wave) (article == def) (number plur))) ))
(pp x)
(unify x grammar)
% ------------------------------------------------------------------------
% 3 : VOICE NONE, PROT ANY : (a diaphragm compresses the waves)
(setq x
  '((cat s)
    (verb ((v == compress)))
    (prot ((n == diaphragm) (article == indef)))
    (goal ((n == wave) (article == def) (number plur))) ))
(pp x)
(unify x grammar)
% ------------------------------------------------------------------------
% 4 : VOICE PASSIVE, PROT ANY : (the waves are compressed by a diaphragm)
(reclaim)
(setq x
  '((cat s)
    (verb ((v == compress) (voice passive)))
    (prot ((n == diaphragm) (article == indef)))
    (goal ((n == wave) (article == def) (number plur))) ))
(pp x)
(unify x grammar)
% ------------------------------------------------------------------------
% 5 : VOICE NONE, PROT NONE : (the waves are compressed)
(setq x
  '((cat s)
    (verb ((v == compress)))
    (goal ((n == wave) (article == def) (number plur))) ))
(pp x)
(unify x grammar)
% ------------------------------------------------------------------------
% 6 : VOICE PASSIVE, PROT NONE : (the waves are compressed)
(setq x
  '((cat s)
    (verb ((v == compress) (voice passive)))
    (goal ((n == wave) (article == def) (number plur))) ))
(pp x)
(unify x grammar)
% ------------------------------------------------------------------------
```

```
% TL-TEST-2.LSP : subordinates in adverbial phrase position
% all 4x4 combinations : 1=prot/2=goal/3=prot+goal/4=prot+goal+passive
% ------------------------------------------------------------------------
%
% 1.1
(setq x
 '((cat s) (order pre)
   (s ((verb ((v == compress) ))
       (prot ((n == transmitter) (article == def)))))
   (subord ((relpro == because)
            (s ((verb ((v == move) ))
                (prot ((n == wave) (article == def)))))))))
(pp x)
(unify x grammar)
% ------------------------------------------------------------------------
% 1.2
(setq x
 '((cat s) (order post)
   (s ((verb ((v == compress) ))
       (prot ((n == transmitter) (article == def)))))
   (subord ((relpro == because)
            (s ((verb ((v == move) ))
                (goal ((n == molecule) (number plur) (article == def)))))))))
(pp x)
(unify x grammar)
% ------------------------------------------------------------------------
% 1.3
(setq x
 '((cat s) (order pre)
   (s ((verb ((v == compress) ))
       (prot ((n == transmitter) (article == def)))))
   (subord ((relpro == because)
            (s ((verb ((v == move) ))
                (prot ((n == wave) (article == def)))
                (goal ((n == molecule) (number plur) (article == def)))))))))
(pp x)
(unify x grammar)
% ------------------------------------------------------------------------
% 1.4
(setq x
 '((cat s)
   (s ((verb ((v == compress) ))
       (prot ((n == transmitter) (article == def)))))
   (subord ((relpro == because)
            (s ((verb ((v == move) (voice passive)))
                (prot ((n == wave) (article == def)))
                (goal ((n == molecule) (number plur) (article == def)))))))))
(pp x)
(unify x grammar)
% ------------------------------------------------------------------------
% 2.1
(setq x
 '((cat s) (order post)
   (s ((verb ((v == compress) ))
       (goal ((n == granule) (number plur) (article == def)))))
   (subord ((relpro == because)
            (s ((verb ((v == move) ))
                (prot ((n == wave) (article == def)))))))))
(pp x)
(unify x grammar)
% ------------------------------------------------------------------------
% 2.2
(setq x
 '((cat s) (order pre)
   (s ((verb ((v == compress) ))
       (goal ((n == granule) (number plur) (article == def)))))
   (subord ((relpro == because)
```

```
            (s ((verb ((v == move) ))
                (goal ((n == molecule) (number plur) (article == def))))))))))
(pp x)
(unify x grammar)
% ----------------------------------------------------------------------
% 2.3
(setq x
 '((cat s)
   (s ((verb ((v == compress) ))
       (goal ((n == granule) (number plur) (article == def)))))
   (subord ((relpro == because)
            (s ((verb ((v == move) ))
                (prot ((n == wave) (article == def)))
                (goal ((n == molecule) (number plur) (article == def))))))))))
(pp x)
(unify x grammar)
% ----------------------------------------------------------------------
% 2.4
(setq x
 '((cat s) (order pre)
   (s ((verb ((v == compress) ))
       (goal ((n == granule) (number plur) (article == def)))))
   (subord ((relpro == because)
            (s ((verb ((v == move) (voice passive)))
                (prot ((n == wave) (article == def)))
                (goal ((n == molecule) (number plur) (article == def))))))))))
(pp x)
(unify x grammar)
% ----------------------------------------------------------------------
% 3.1
(setq x
 '((cat s) (order pre)
   (s ((verb ((v == compress) ))
       (prot ((n == transmitter) (article == def)))
       (goal ((n == granule) (number plur) (article == def)))))
   (subord ((relpro == because)
            (s ((verb ((v == move) ))
                (prot ((n == wave) (article == def))))))))))
(pp x)
(unify x grammar)
% ----------------------------------------------------------------------
% 3.2
(setq x
 '((cat s) (order post)
   (s ((verb ((v == compress) ))
       (prot ((n == transmitter) (article == def)))
       (goal ((n == granule) (number plur) (article == def)))))
   (subord ((relpro == because)
            (s ((verb ((v == move) ))
                (goal ((n == molecule) (number plur) (article == def))))))))))
(pp x)
(unify x grammar)
% ----------------------------------------------------------------------
% 3.3
(setq x
 '((cat s) (order pre)
   (s ((verb ((v == compress) ))
       (prot ((n == transmitter) (article == def)))
       (goal ((n == granule) (number plur) (article == def)))))
   (subord ((relpro == because)
            (s ((verb ((v == move) ))
                (prot ((n == wave) (article == def)))
                (goal ((n == molecule) (number plur) (article == def))))))))))
(pp x)
(unify x grammar)
% ----------------------------------------------------------------------
```

```
% 3.4
(setq x
  '((cat s)
    (s ((verb ((v == compress) ))
        (prot ((n == transmitter) (article == def)))
        (goal ((n == granule) (number plur) (article == def)))))
    (subord ((relpro == because)
             (s ((verb ((v == move) (voice passive)))
                 (prot ((n == wave) (article == def)))
                 (goal ((n == molecule) (number plur) (article == def))))))))))
(pp x)
(unify x grammar)
% ----------------------------------------------------------------------
% 4.1
(setq x
  '((cat s) (order post)
    (s ((verb ((v == compress) (voice passive)))
        (prot ((n == transmitter) (article == def)))
        (goal ((n == granule) (number plur) (article == def)))))
    (subord ((relpro == because)
             (s ((verb ((v == move) ))
                 (prot ((n == wave) (article == def)))))))))
(pp x)
(unify x grammar)
% ----------------------------------------------------------------------
% 4.2
(setq x
  '((cat s) (order pre)
    (s ((verb ((v == compress) (voice passive)))
        (prot ((n == transmitter) (article == def)))
        (goal ((n == granule) (number plur) (article == def)))))
    (subord ((relpro == because)
             (s ((verb ((v == move) ))
                 (goal ((n == molecule) (number plur) (article == def))))))))
(pp x)
(unify x grammar)
% ----------------------------------------------------------------------
% 4.3
(setq x
  '((cat s) (order pre)
    (s ((verb ((v == compress) (voice passive)))
        (prot ((n == transmitter) (article == def)))
        (goal ((n == granule) (number plur) (article == def)))))
    (subord ((relpro == because)
             (s ((verb ((v == move) ))
                 (prot ((n == wave) (article == def)))
                 (goal ((n == molecule) (number plur) (article == def))))))))
(pp x)
(unify x grammar)
% ----------------------------------------------------------------------
% 4.4
(setq x
  '((cat s)
    (s ((verb ((v == compress) (voice passive)))
        (prot ((n == transmitter) (article == def)))
        (goal ((n == granule) (number plur) (article == def)))))
    (subord ((relpro == because)
             (s ((verb ((v == move) (voice passive)))
                 (prot ((n == wave) (article == def)))
                 (goal ((n == molecule) (number plur) (article == def))))))))
(pp x)
(unify x grammar)
% ----------------------------------------------------------------------
```

```
% TL-TEST-3.LSP : part of TL-TEST-2.LSP less specified : S in SUBORD omitted
% particularly tested : variation in number and article of both protagonists
% ------------------------------------------------------------------------
(setq x
  '((cat s) (order pre) (verb ((v == compress)))
    (prot ((n == transmitter) (article == def)))
    (subord ((relpro == because) (verb ((v == move)))
             (prot ((n == wave) (article == def)))))))
(pp x)
(unify x grammar)
% (because the wave moves, the transmitter compresses)
% ------------------------------------------------------------------------
(setq x
  '((cat s) (order pre) (verb ((v == compress)))
    (prot ((n == transmitter) (article == def) (number plur)))
    (subord ((relpro == because) (verb ((v == move)))
             (prot ((n == wave) (article == def)))))))
(pp x)
(unify x grammar)
% (because the wave moves, the transmitters compress)
% ------------------------------------------------------------------------
(setq x
  '((cat s) (order pre) (verb ((v == compress)))
    (prot ((n == transmitter) (article == def) (number plur)))
    (subord ((relpro == because) (verb ((v == move)))
             (prot ((n == wave) (article == indef)))))))
(pp x)
(unify x grammar)
% (because a wave moves, the transmitters compress)
% ------------------------------------------------------------------------
(setq x
  '((cat s) (order pre) (verb ((v == compress)))
    (prot ((n == transmitter) (article == def)))
    (subord ((relpro == because) (verb ((v == move)))
             (prot ((n == wave) (article == def) (number plur)))))))
(pp x)
(unify x grammar)
% (because the waves move, the transmitter compresses)
% ------------------------------------------------------------------------
(setq x
  '((cat s) (order pre) (verb ((v == compress)))
    (prot ((n == transmitter) (article == indef)))
    (subord ((relpro == because) (verb ((v == move)))
             (prot ((n == wave) (article == def) (number plur)))))))
(pp x)
(unify x grammar)
% (because the waves move, a transmitter compresses)
% ------------------------------------------------------------------------
(setq x
  '((cat s) (order pre) (verb ((v == compress)))
    (prot ((n == transmitter) (article == indef)))
    (subord ((relpro == because) (verb ((v == move)))
             (prot ((n == wave) (article == indef) (number plur)))))))
(pp x)
(unify x grammar)
% (because waves move, a transmitter compresses)
% ------------------------------------------------------------------------
```

```
% TL-TEST-4.LSP : embed, aspect; subord, aspect
% ------------------------------------------------------------------------
(setq x
  '((cat s)
    (verb ((v == cause)))
    (prot ((embed ((relpro == that)
                   (verb ((v == speak)))
                   (prot ((n == person) (article == indef)))))))
    (goal ((embed ((relpro == that)
                   (verb ((v == hit)))
                   (prot ((n == wave) (number plur)))
                   (goal ((n == diaphragm) (article == def)))))))))
(pp x)
(unify x grammar)
% (that a person speaks causes that waves hit the diaphragm)
% ------------------------------------------------------------------------
(setq x
  '((cat s)
    (verb ((v == cause)))
    (prot ((embed ((relpro == that)
                   (verb ((v == speak)))
                   (prot ((n == person) (article == indef)))))))
    (goal ((embed ((verb ((v == hit) (aspect inf)))
                   (prot ((n == wave) (number plur)))
                   (goal ((n == diaphragm) (article == def)))))))))
(pp x)
(unify x grammar)
% (that a person speaks causes waves to hit the diaphragm)
% ========================================================================
(setq x
  '((cat s) (order pre)
    (verb ((v == compress)))
    (prot ((n == diaphragm) (article == def)))
    (goal ((n == granule) (article == def) (number plur)))
    (subord ((relpro == because)
             (verb ((v == move)))
             (prot ((n == wave) (number plur) (article == def)))
             (goal ((n == molecule) (article == def) (number plur))))))))
(pp x)
(unify x grammar)
% (because the waves move the molecules, the diaphragm compresses the granules
% ------------------------------------------------------------------------
(setq x
  '((cat s) (order pre)
    (verb ((v == compress)))
    (prot ((n == diaphragm) (article == def)))
    (goal ((n == granule) (article == def) (number plur)))
    (subord ((verb ((v == move) (aspect ing)))
             (prot ((n == wave) (number plur) (article == def)))
             (goal ((n == molecule) (number plur) (article == def))))))))
(pp x)
(unify x grammar)
% (the waves moving the molecules, the diaphragm compresses the granules)
% ========================================================================
```

```
% TL-TEST-5.LSP : 4 possible combinations for relative clause:
% active, prot-subj relativized
% active, goal-dobj relativized
% passive, goal-subj relativized
% passive, prot-by-obj relativized
% ------------------------------------------------------------------------
(setq x
  '((cat s-bar) (relpro == that)
    (s ((verb ((v == compress))) (prot ((gap !+)))
        (goal ((n == diaphragm) (article == indef)))))))
(pp x)
(unify x grammar)
% active, prot-subject relativized
% (that compresses a diaphragm)
% ------------------------------------------------------------------------
(setq x
  '((cat s-bar) (relpro == that)
    (s ((verb ((v == compress)))
        (prot ((n == diaphragm) (article == indef)))
        (goal ((gap !+)))))))
(pp x)
(unify x grammar)
% active, goal-object relativized
% (that a diaphragm compresses)
% ------------------------------------------------------------------------
(setq x
  '((cat s-bar) (relpro == that)
    (s ((verb ((v == compress) (voice passive)))
        (prot ((n == diaphragm) (article == indef)))
        (goal ((gap !+)))))))
(pp x)
(unify x grammar)
% passive, goal-subject relativized
% (that is compressed by a diaphragm)
% ------------------------------------------------------------------------
(setq x
  '((cat s-bar) (relpro == that)
    (s ((verb ((v == compress) (voice passive))) (prot ((gap !+)))
        (goal ((n == diaphragm) (article == indef)))))))
(pp x)
(unify x grammar)
% passive, prot-by-obj relativized
% (that a diaphragm is compressed by)
% ------------------------------------------------------------------------
(setq x
  '((cat np) (n == wave)
    (tail ((relpro == that)
           (s ((verb ((v == compress))) (prot ((gap !+)))
               (goal ((n == diaphragm) (article == indef)))))))))
(pp x)
(unify x grammar)
% antecedent : sing
% (wave that compresses a diaphragm)
% ------------------------------------------------------------------------
(setq x
  '((cat np) (n == wave) (number plur)
    (tail ((relpro == that)
           (s ((verb ((v == compress))) (prot ((gap !+)))
               (goal ((n == diaphragm) (article == indef)))))))))
(pp x)
(unify x grammar)
% antecedent : plur
% (waves that compress a diaphragm)
% ------------------------------------------------------------------------
```

```
TL-TEST-5-OUT.LSP  5 Aug 1985 1445-EDT

4 possible combinations for relative clause :
active , prot-subj relativized
active , goal-dobj relativized
passive , goal-subj relativized
passive , prot-by-obj relativized

-------------------------------------------------------------------------
(SETQ X '((CAT S-BAR) (RELPRO == THAT)
          (S ((VERB ((V == COMPRESS))) (PROT ((GAP !+)))
              (GOAL ((N == DIAPHRAGM) (ARTICLE == INDEF)))))))
(THAT COMPRESSES A DIAPHRAGM!.)
active , prot-subject relativized
-------------------------------------------------------------------------
(SETQ X '((CAT S-BAR) (RELPRO == THAT)
          (S ((VERB ((V == COMPRESS)))
              (PROT ((N == DIAPHRAGM) (ARTICLE == INDEF)))
              (GOAL ((GAP !+)))))))
(THAT A DIAPHRAGM COMPRESSES!.)
active , goal-object relativized
-------------------------------------------------------------------------
(SETQ X '((CAT S-BAR) (RELPRO == THAT)
          (S ((VERB ((V == COMPRESS) (VOICE PASSIVE)))
              (PROT ((N == DIAPHRAGM) (ARTICLE == INDEF)))
              (GOAL ((GAP !+)))))))
(THAT IS COMPRESSED BY A DIAPHRAGM!.)
passive , goal-subject relativized
-------------------------------------------------------------------------
(SETQ X '((CAT S-BAR) (RELPRO == THAT)
          (S ((VERB ((V == COMPRESS) (VOICE PASSIVE))) (PROT ((GAP !+)))
              (GOAL ((N == DIAPHRAGM) (ARTICLE == INDEF)))))))
FAILURE
NIL
passive , prot-by-obj relativized ( = that a diaphragm is compressed by )
-------------------------------------------------------------------------

number agreement ? - test :

-------------------------------------------------------------------------
(SETQ X '((CAT NP) (N == WAVE)
          (TAIL ((RELPRO == THAT)
                 (S ((VERB ((V == COMPRESS))) (PROT ((GAP !+)))
                     (GOAL ((N == DIAPHRAGM) (ARTICLE == INDEF)))))))))
(TAVE THAT COMPRESSES A DIAPHRAGM!.)
-------------------------------------------------------------------------
(SETQ X '((CAT NP) (N == WAVE) (NUMBER PLUR)
          (TAIL ((RELPRO == THAT)
                 (S ((VERB ((V == COMPRESS))) (PROT ((GAP !+)))
                     (GOAL ((N == DIAPHRAGM) (ARTICLE == INDEF)))))))))
(TAVES THAT COMPRESSES DIAPHRAGM!.)
-------------------------------------------------------------------------

no number agreement
( indeed not possible , since there is no reference to the antecedent )

NB : also a subsidiary problem of vanishing article INDEF !
```

Owner	KWEE
Name	PS:<KWEE>TL-TEST-FINALOUT
[Date]	20-Aug-85 4:30:05PM
Jobheader	
PageCollation	
FormLength	66
Language	Printer

20-Aug-85 02:45:06

TL-TEST-1.OUT ----------

```
(SETQ X
      '((CAT S) (VERB ((V == COMPRESS) (VOICE ACTIVE)))
        (GOAL ((N == WAVE) (ARTICLE == DEF) (NUMBER PLUR)))))
1: NIL

(SETQ X
      '((CAT S) (VERB ((V == COMPRESS) (VOICE ACTIVE)))
        (PROT ((N == DIAPHRAGM) (ARTICLE == INDEF)))
        (GOAL ((N == WAVE) (ARTICLE == DEF) (NUMBER PLUR)))))
2: (A DIAPHRAGM COMPRESSES THE WAVES!.)

(SETQ X
      '((CAT S) (VERB ((V == COMPRESS)))
        (PROT ((N == DIAPHRAGM) (ARTICLE == INDEF)))
        (GOAL ((N == WAVE) (ARTICLE == DEF) (NUMBER PLUR)))))
3: (A DIAPHRAGM COMPRESSES THE WAVES!.)

(SETQ X
      '((CAT S) (VERB ((V == COMPRESS) (VOICE PASSIVE)))
        (PROT ((N == DIAPHRAGM) (ARTICLE == INDEF)))
        (GOAL ((N == WAVE) (ARTICLE == DEF) (NUMBER PLUR)))))
4: (THE WAVES ARE COMPRESSED BY A DIAPHRAGM!.)

(SETQ X
      '((CAT S) (VERB ((V == COMPRESS)))
        (GOAL ((N == WAVE) (ARTICLE == DEF) (NUMBER PLUR)))))
5: (THE WAVES ARE COMPRESSED!.)

(SETQ X
      '((CAT S) (VERB ((V == COMPRESS) (VOICE PASSIVE)))
        (GOAL ((N == WAVE) (ARTICLE == DEF) (NUMBER PLUR)))))
6: (THE WAVES ARE COMPRESSED!.)
```

20-Aug-85 02:45:06

TL-TEST-2.OUT ----------

```
(SETQ X
      '((CAT S) (ORDER PRE)
        (S ((VERB ((V == COMPRESS)))
            (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE)))
                     (PROT ((N == WAVE) (ARTICLE == DEF)))))))))
(BECAUSE THE WAVE MOVES!, THE TRANSMITTER COMPRESSES!.)

(SETQ X
      '((CAT S) (ORDER POST)
        (S ((VERB ((V == COMPRESS)))
            (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE)))
                     (GOAL
                      ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
(THE TRANSMITTER COMPRESSES!, BECAUSE THE MOLECULES ARE MOVED!.)

(SETQ X
      '((CAT S) (ORDER PRE)
        (S ((VERB ((V == COMPRESS)))
            (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))))
```

```
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE)))
                     (PROT ((N == WAVE) (ARTICLE == DEF)))
                     (GOAL
                      ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
(BECAUSE THE WAVE MOVES THE MOLECULES!, THE TRANSMITTER COMPRESSES!.)

(SETQ X
      '((CAT S)
        (S ((VERB ((V == COMPRESS)))
            (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE) (VOICE PASSIVE)))
                     (PROT ((N == WAVE) (ARTICLE == DEF)))
                     (GOAL
                      ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
(THE TRANSMITTER COMPRESSES!, BECAUSE THE MOLECULES ARE MOVED BY THE WAVE!.)

(SETQ X
      '((CAT S) (ORDER POST)
        (S ((VERB ((V == COMPRESS)))
            (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE)))
                     (PROT ((N == WAVE) (ARTICLE == DEF)))))))))
(THE GRANULES ARE COMPRESSED!, BECAUSE THE WAVE MOVES!.)

(SETQ X
      '((CAT S) (ORDER PRE)
        (S ((VERB ((V == COMPRESS)))
            (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE)))
                     (GOAL
                      ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
(BECAUSE THE MOLECULES ARE MOVED!, THE GRANULES ARE COMPRESSED!.)

(SETQ X
      '((CAT S)
        (S ((VERB ((V == COMPRESS)))
            (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE)))
                     (PROT ((N == WAVE) (ARTICLE == DEF)))
                     (GOAL
                      ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
(THE GRANULES ARE COMPRESSED!, BECAUSE THE WAVE MOVES THE MOLECULES!.)

(SETQ X
      '((CAT S) (ORDER PRE)
        (S ((VERB ((V == COMPRESS)))
            (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE) (VOICE PASSIVE)))
                     (PROT ((N == WAVE) (ARTICLE == DEF)))
                     (GOAL
                      ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
(BECAUSE THE MOLECULES ARE MOVED BY THE WAVE!, THE GRANULES ARE COMPRESSED!.)

(SETQ X
      '((CAT S) (ORDER PRE)
        (S ((VERB ((V == COMPRESS)))
            (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))
            (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE)))
```

```
                        (PROT ((N == WAVE) (ARTICLE == DEF)))))))))
 (BECAUSE THE WAVE MOVES!, THE TRANSMITTER COMPRESSES THE GRANULES!.)

 (SETQ X
       '((CAT S) (ORDER POST)
         (S ((VERB ((V == COMPRESS)))
             (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))
             (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
         (SUBORD ((RELPRO == BECAUSE)
                  (S ((VERB ((V == MOVE)))
                      (GOAL
                       ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
 (THE TRANSMITTER COMPRESSES THE GRANULES!, BECAUSE THE MOLECULES ARE MOVED!.)

 (SETQ X
       '((CAT S) (ORDER PRE)
         (S ((VERB ((V == COMPRESS)))
             (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))
             (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
         (SUBORD ((RELPRO == BECAUSE)
                  (S ((VERB ((V == MOVE)))
                      (PROT ((N == WAVE) (ARTICLE == DEF)))
                      (GOAL
                       ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
(BECAUSE THE WAVE MOVES THE MOLECULES!, THE TRANSMITTER COMPRESSES THE
GRANULES!.)

(SETQ X
      '((CAT S)
        (S ((VERB ((V == COMPRESS)))
            (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))
            (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE) (VOICE PASSIVE)))
                     (PROT ((N == WAVE) (ARTICLE == DEF)))
                     (GOAL
                      ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
(THE TRANSMITTER COMPRESSES THE GRANULES!, BECAUSE THE MOLECULES ARE MOVED
BY THE WAVE!.)

(SETQ X
      '((CAT S) (ORDER POST)
        (S ((VERB ((V == COMPRESS) (VOICE PASSIVE)))
            (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))
            (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE)))
                     (PROT ((N == WAVE) (ARTICLE == DEF)))))))))
(THE GRANULES ARE COMPRESSED BY THE TRANSMITTER!, BECAUSE THE WAVE MOVES!.)

(SETQ X
      '((CAT S) (ORDER PRE)
        (S ((VERB ((V == COMPRESS) (VOICE PASSIVE)))
            (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))
            (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE)))
                     (GOAL
                      ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
(BECAUSE THE MOLECULES ARE MOVED!, THE GRANULES ARE COMPRESSED BY THE
TRANSMITTER!.)

(SETQ X
      '((CAT S) (ORDER PRE)
        (S ((VERB ((V == COMPRESS) (VOICE PASSIVE)))
            (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))
```

```
            (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE)))
                     (PROT ((N == WAVE) (ARTICLE == DEF)))
                     (GOAL
                      ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
(BECAUSE THE WAVE MOVES THE MOLECULES!, THE GRANULES ARE COMPRESSED BY THE
TRANSMITTER!.)

(SETQ X
      '((CAT S)
        (S ((VERB ((V == COMPRESS) (VOICE PASSIVE)))
            (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))
            (GOAL ((N == GRANULE) (NUMBER PLUR) (ARTICLE == DEF)))))
        (SUBORD ((RELPRO == BECAUSE)
                 (S ((VERB ((V == MOVE) (VOICE PASSIVE)))
                     (PROT ((N == WAVE) (ARTICLE == DEF)))
                     (GOAL
                      ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))))
(THE GRANULES ARE COMPRESSED BY THE TRANSMITTER!, BECAUSE THE MOLECULES ARE
MOVED BY THE WAVE!.)


20-Aug-85 02:45:06

TL-TEST-3.OUT ----------------------------------------------------------

(SETQ X
      '((CAT S) (ORDER PRE) (VERB ((V == COMPRESS)))
        (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))
        (SUBORD ((RELPRO == BECAUSE) (VERB ((V == MOVE)))
                 (PROT ((N == WAVE) (ARTICLE == DEF))))))))
(BECAUSE THE WAVE MOVES!, THE TRANSMITTER COMPRESSES!.)

(SETQ X
      '((CAT S) (ORDER PRE) (VERB ((V == COMPRESS)))
        (PROT ((N == TRANSMITTER) (ARTICLE == DEF) (NUMBER PLUR)))
        (SUBORD ((RELPRO == BECAUSE) (VERB ((V == MOVE)))
                 (PROT ((N == WAVE) (ARTICLE == DEF))))))))
(BECAUSE THE WAVE MOVES!, THE TRANSMITTERS COMPRESS!.)

(SETQ X
      '((CAT S) (ORDER PRE) (VERB ((V == COMPRESS)))
        (PROT ((N == TRANSMITTER) (ARTICLE == DEF) (NUMBER PLUR)))
        (SUBORD ((RELPRO == BECAUSE) (VERB ((V == MOVE)))
                 (PROT ((N == WAVE) (ARTICLE == INDEF))))))))
(BECAUSE A WAVE MOVES!, THE TRANSMITTERS COMPRESS!.)

(SETQ X
      '((CAT S) (ORDER PRE) (VERB ((V == COMPRESS)))
        (PROT ((N == TRANSMITTER) (ARTICLE == DEF)))
        (SUBORD ((RELPRO == BECAUSE) (VERB ((V == MOVE)))
                 (PROT ((N == WAVE) (ARTICLE == DEF) (NUMBER PLUR)))))))
(BECAUSE THE WAVES MOVE!, THE TRANSMITTER COMPRESSES!.)

(SETQ X
      '((CAT S) (ORDER PRE) (VERB ((V == COMPRESS)))
        (PROT ((N == TRANSMITTER) (ARTICLE == INDEF)))
        (SUBORD ((RELPRO == BECAUSE) (VERB ((V == MOVE)))
                 (PROT ((N == WAVE) (ARTICLE == DEF) (NUMBER PLUR)))))))
(BECAUSE THE WAVES MOVE!, A TRANSMITTER COMPRESSES!.)

(SETQ X
      '((CAT S) (ORDER PRE) (VERB ((V == COMPRESS)))
        (PROT ((N == TRANSMITTER) (ARTICLE == INDEF)))
        (SUBORD ((RELPRO == BECAUSE) (VERB ((V == MOVE)))
```

```
                    (PROT ((N == WAVE) (ARTICLE == INDEF) (NUMBER PLUR)))))))
(BECAUSE WAVES MOVE!, A TRANSMITTER COMPRESSES!.)


20-Aug-85 02:45:06

TL-TEST-4.OUT ----------------------------------------------------------

(SETQ X
      '((CAT S) (VERB ((V == CAUSE)))
        (PROT ((EMBED ((RELPRO == THAT) (VERB ((V == SPEAK)))
                       (PROT ((N == PERSON) (ARTICLE == INDEF)))))))
        (GOAL ((EMBED ((RELPRO == THAT) (VERB ((V == HIT)))
                       (PROT ((N == WAVE) (NUMBER PLUR)))
                       (GOAL ((N == DIAPHRAGM) (ARTICLE == DEF)))))))))
(THAT A PERSON SPEAKS CAUSES THAT WAVES HIT THE DIAPHRAGM!.)

(SETQ X
      '((CAT S) (VERB ((V == CAUSE)))
        (PROT ((EMBED ((RELPRO == THAT) (VERB ((V == SPEAK)))
                       (PROT ((N == PERSON) (ARTICLE == INDEF)))))))
        (GOAL ((EMBED ((VERB ((V == HIT) (ASPECT INF)))
                       (PROT ((N == WAVE) (NUMBER PLUR)))
                       (GOAL ((N == DIAPHRAGM) (ARTICLE == DEF)))))))))
(THAT A PERSON SPEAKS CAUSES WAVES TO HIT THE DIAPHRAGM!.)

(SETQ X
      '((CAT S) (ORDER PRE) (VERB ((V == COMPRESS)))
        (PROT ((N == DIAPHRAGM) (ARTICLE == DEF)))
        (GOAL ((N == GRANULE) (ARTICLE == DEF) (NUMBER PLUR)))
        (SUBORD ((RELPRO == BECAUSE) (VERB ((V == MOVE)))
                 (PROT ((N == WAVE) (NUMBER PLUR) (ARTICLE == DEF)))
                 (GOAL ((N == MOLECULE) (ARTICLE == DEF) (NUMBER PLUR)))))))
(BECAUSE THE WAVES MOVE THE MOLECULES!, THE DIAPHRAGM COMPRESSES THE
GRANULES!.)

(SETQ X
      '((CAT S) (ORDER PRE) (VERB ((V == COMPRESS)))
        (PROT ((N == DIAPHRAGM) (ARTICLE == DEF)))
        (GOAL ((N == GRANULE) (ARTICLE == DEF) (NUMBER PLUR)))
        (SUBORD ((VERB ((V == MOVE) (ASPECT ING)))
                 (PROT ((N == WAVE) (NUMBER PLUR) (ARTICLE == DEF)))
                 (GOAL ((N == MOLECULE) (NUMBER PLUR) (ARTICLE == DEF)))))))
(THE WAVES MOVE-ING THE MOLECULES!, THE DIAPHRAGM COMPRESSES THE GRANULES!.)


20-Aug-85 02:45:06

TL-TEST-5.OUT ----------------------------------------------------------

(SETQ X
      '((CAT S-BAR) (RELPRO == THAT)
        (S ((VERB ((V == COMPRESS))) (PROT ((GAP !+)))
            (GOAL ((N == DIAPHRAGM) (ARTICLE == INDEF)))))))
(THAT COMPRESSES A DIAPHRAGM!.)

(SETQ X
      '((CAT S-BAR) (RELPRO == THAT)
        (S ((VERB ((V == COMPRESS)))
            (PROT ((N == DIAPHRAGM) (ARTICLE == INDEF)))
            (GOAL ((GAP !+)))))))
(THAT A DIAPHRAGM COMPRESSES!.)

(SETQ X
      '((CAT S-BAR) (RELPRO == THAT)
        (S ((VERB ((V == COMPRESS) (VOICE PASSIVE)))
```

```
            (PROT ((N == DIAPHRAGM) (ARTICLE == INDEF)))
            (GOAL ((GAP !+)))))))
(THAT IS COMPRESSED BY A DIAPHRAGM!.)

(SETQ X
      '((CAT S-BAR) (RELPRO == THAT)
        (S ((VERB ((V == COMPRESS) (VOICE PASSIVE))) (PROT ((GAP !+)))
            (GOAL ((N == DIAPHRAGM) (ARTICLE == INDEF)))))))
NIL

FAILURE ON one of the inside constituent

Constituent being processed:
(SETQ CON 'BY-OBJ)

(SETQ CONVAL '((CAT PP) (PREP ((LEX BY))) (NP (^ PROT))))
 With Input:
(SETQ I
      '((CSET (GOAL VERB VERB BY-OBJ)) (SUBLIST NONE)
        (SENTENCE-ADVERBIAL NONE) (PATTERN (GOAL VERB VERB BY-OBJ POUND))
        (VERB ((NUMBER NONE) (CAT VERB-GROUP) (VOICE PASSIVE)
               (V ((LEX COMPRESS)))))
        (BENEF NONE)
        (GOAL ((GAP NONE) (CAT NP) (N ((LEX DIAPHRAGM)))
               (ARTICLE ((LEX INDEF)))))
        (BY-OBJ ((CAT PP) (PREP ((LEX BY))) (NP (^ PROT))))
        (PROT ((GAP !+) (CAT GAP))) (SUBORD NONE) (CONJ NONE) (CAT S)))

FAILURE ON one of the inside constituent

Constituent being processed:
(SETQ CON 'S)

(SETQ CONVAL
      '((CAT S) (VERB ((V ((LEX COMPRESS))) (VOICE PASSIVE)))
        (PROT ((GAP !+)))
        (GOAL ((N ((LEX DIAPHRAGM))) (ARTICLE ((LEX INDEF)))))))
 With Input:
(SETQ I
      '((CSET (RELPRO S)) (PATTERN (RELPRO S))
        (RELPRO ((LEX THAT) (CAT RELPRO)))
        (S ((CAT S) (VERB ((V ((LEX COMPRESS))) (VOICE PASSIVE)))
            (PROT ((GAP !+)))
            (GOAL ((N ((LEX DIAPHRAGM))) (ARTICLE ((LEX INDEF)))))))
        (CAT S-BAR)))
NIL

(SETQ X
      '((CAT NP) (N == WAVE)
        (TAIL ((RELPRO == THAT)
               (S ((VERB ((V == COMPRESS))) (PROT ((GAP !+)))
                   (GOAL ((N == DIAPHRAGM) (ARTICLE == INDEF)))))))))
(TAVE THAT COMPRESSES A DIAPHRAGM!.)

(SETQ X
      '((CAT NP) (N == WAVE) (NUMBER PLUR)
        (TAIL ((RELPRO == THAT)
               (S ((VERB ((V == COMPRESS))) (PROT ((GAP !+)))
                   (GOAL ((N == DIAPHRAGM) (ARTICLE == INDEF)))))))))
(TAVES THAT COMPRESSES DIAPHRAGM!.)
```

Owner	KWEE
Name	PS:<KWEE>TL-LINEAR-4PLUS.L
[Date]	18-Aug-85 2:09:31PM
Jobheader	
PageCollation	
FormLength	66
Language	Printer

```
% FILE <KWEE>TL-LINEAR-4PLUS.LSP - 23 May 1923-EDT - redef 4 functions
% for: TO + infinitive, and ING-forms; as well as better 3rd sg. morphology;;
% and also : 30 Jul 1985 1618-EDT : Cecile's modification in punctuate;
% modifications in copy 21 May 1985 0937-EDT of <MCKEOWN.VAX>LINEAR.L



% This is the redefinition of the function Punctuate.
% The only change, however, is 1 line (as indicated by the comment):
% where you have the period ".", replace it by (int2id (car (string2list ".")))
% and do similarly for the comma.
% We have not seen quotes yet, or other punctuation.
% But the same problem may happen for those. -- Cecile

(def punctuate
  (lambda (sentence)
  %    **)
  %    ** **********************************)
  %    **)
  %    ** Concatenates any punctuatation with the word)
  %    ** it precedes --- punctuation in the sentence should)
  %    ** look like (punctuation ".") (e-g-) --- The last word)
  %    ** is given a period ------------------------------)
  %    **)
  %    ** **********************************)
  %    **)

% July 5, 1985: Cecile's change: to get the comma and period.
% As these are special characters, they need to be transformed first
% into ascii code, then back to an identifier.
% Basically, the period and the comma have to be replaced by:
%         (int2id (car (string2list ".")))
%         (int2id (car (string2list ",")))
% May be the same will happen for the quote????

    (cond ((null (cdr sentence))
           (list (atom-concat  (car sentence)
                               (int2id (car (string2list "."))))))
          ((atom (cadr sentence))
           (cons (car sentence) (punctuate (cdr sentence))))
          ((and (equal (cadadr sentence) (int2id (car (string2list ","))))
                (member (caddr sentence) '(and or through)))
           (cons (car sentence) (punctuate (cddr sentence))))
          (t
           (punctuate
            (cons (atom-concat  (car sentence)
                                (int2id (car (string2list (cadadr sentence))))
                  (cddr sentence)))))))
```

```
(def morph
  (lambda (cat lex fd fd-stack)
%    **)
%    ** ****************************************)
%    **)
%    ** Returns: the word or list of words represented)
%    ** by fd ---- The morphological processing of the)
%    ** word depends upon its category ---- a single lexical)
%    ** item is expanded into several words if it contains @)
%    **)
%    ** ******************************************)
%    **)
   (remove-@
    (explode
     (selectq cat
              (adj lex)
              (conj lex)
              (noun
               (morph-noun lex
                           (retrieve-from 'number
                                          (cons fd fd-stack)
                                          '(cat np))
                           (retrieve 'feature fd fd-stack 1)
                           cat))
              (adv lex)
              (prep lex)
              (verb
               (morph-verb lex
% ************************************************************ aspect : TL
                           (retrieve 'aspect fd fd-stack nil)
                           (retrieve 'number fd fd-stack nil)
                           (retrieve 'person fd fd-stack nil)
                           (retrieve 'tense fd fd-stack nil)
                           (lex-entry cat lex (lexicon grammar))))
              (article
               (morph-article lex
                              (next-word (retrieve 'pattern
                                                   fd
                                                   fd-stack
                                                   nil)
                                         fd
                                         fd-stack)
                              (retrieve 'number fd fd-stack nil)))
              (relpro lex)
              (s-adv lex)
              (punctuation lex)
              (nplist nil)
              (slist nil)
              (pplist nil)
              (gap nil)
              (msg (N 1) "Illegal category" (B 3) cat (N 1)))))))


(def morph-verb
  (lambda (word aspect number person tense entry)
% ************************************************************ aspect : TL
%    **)
%    ** ********************************************)
%    **)
%    ** Adds the proper suffix to the verb root taking into account)
%    ** number - person - and tense ---- default on person is third)
%    ** -- default on number is singular --- default on tense is present)
%    **)
%    ** ******************************************)
%    **)
```

```
    (cond
% ********************************************************** aspect : TL
          ((or (equal aspect 'inf) (equal aspect 'ing))
           (cond ((equal tense 'pastp)
                  (cond ((irregular-verb entry number person tense))
                        (t (form-past (explodec word) word)) ))
                 ((equal aspect 'ing) (form-ing (explodec word) word))
                 (t (list 'to word)) ) )
% ********************************************************** aspect : TL
          ((irregular-verb entry number person tense))
          ((equal tense 'present)
           (form-present-verb word number person))
          ((equal tense 'past) (form-past (explodec word) word))
          ((equal tense 'pastp) (form-past (explodec word) word))
          ((null tense) (form-present-verb word number person))
          (t nil))))


% ---------------------------------------------------------- aspect : TL
% ********************************************************** aspect : TL
(def form-ing
  (lambda (word-letters word)
    (cond ((not (equal (car (last word-letters)) 'e)) (atom-concat word 'ing))
          (t (atom-concat word '-ing))) ))

% THIS IS DELIBERATELY INCORRECT : THE FINAL 'e HAS TO BE REMOVED ,
% AND THEN 'ing HAS TO BE ADDED : FOR INSTANCE causing, having, moving, etc.
% EXCEPT FOR MONOSYLLABIC (!) VERBS LIKE being, seeing, etc.
% AND DERIVATES, AND OTHERS : LAST SYLLABLE ACCENTUATED : refereeing, etc.
% YET ANOTHER PROBLEM : duplication of final consonant : hitting, running, etc
% THIS APPLIES AS WELL FOR PAST TENSE/PARTICIPLE : plugged, debugged, etc.
% ********************************************************** aspect : TL
% ---------------------------------------------------------- aspect : TL


(def form-present-verb
  (lambda (word number person)
  %    **)
  %    ** **************************************)
  %    **)
  %    ** Forms the suffix for the present tense of the verb word)
  %    **)
  %    ** **************************************)
  %    **)
    (selectq (check-person person)
             (first word)
             (second word)
             (third
              (selectq (check-number number)
                       (sing
% ********************************************************************** TL
                        (cond ((member (car (last (explodec word)))
                                       '(h o s x z))
% ****************************** SCRATCHES, LOATHES, - but : LAUGHS -
% ****************************** DOES, GOES, WOOES, etc.
% ****************************** THIS IS NOT QUITE COMPLETE !
                               (atom-concat word 'es))
                              (t (atom-concat word 's)) ))
% ********************************************************************** TL
                       (plur word)
                       nil))
             nil)))
```

Owner	KWEE
Name	PS:<KWEE>TL-FULLGRAM-8.LS
[Date]	20-Aug-85 12:29:12PM
Jobheader	
PageCollation	
FormLength	66
Language	Printer

```
% TL-FULLGRAM-8.LSP  5 Aug 1985 1038-EDT (Cecile check esp. NP) = TL-GR13 p...
% 1) add in passive: goal none, verb number sing, pattern
% 2) add gap none (2 prot, 2 goal)
% 3) add voice there-insert (in verb-voices and in cat verb-group)
% 4) add s-adv, sublist (in cat s)
% 5) add adv (in cat verb-group)
% 6) add dots in passive prot any pattern verb by-obj
% 7) add prot gap any (twice), mind : (prot ((alt - 2 left parentheses!
% 8) add benef - in active
% 9) add benef - in passive - with verb passive adapted by TL
% 10 factor out nnp in (cat np) under possessive, article
% 11 add under passive: prot goal benef none, verb number sing, pattern
% 12 add goal gap any in active and passive
% 13 subord post (= pattern s subord) with punctuation - 12 Aug 1985 1501-EDT
% 14 take out cat possessive, change lexicon # 12      - 16 Aug 1985 1619-EDT
% 15 factor out nnp in (cat nnp) with adj, pp, tail    - 19 Aug 1985 1603-EDT
%
%
(setq grammarfns '(grammar list-grammar lexicon))
%
%
(setq grammar
  '((alt
% 01 CAT S ------------------------------------------------------------------
     (((cat s)
       (alt
% ------------------------------------------------------------------------
        (((conj none)
          (alt
% ------------------------------------------------------------------------
           (((subord none)
             (alt
% ------------------------------------------------------------------------
%
% VERB VOICE ACTIVE (DEFAULT OR EXPLICIT)
              (((verb ((voice active)))
                (prot any)
                (prot ((alt (((gap none) (cat np))
                             ((gap any) (cat gap) (gap +))))))
                (pattern (prot dots verb dots pound))
                (verb ((cat verb-group)
                       (number (^ prot number))))
% ........................................................................
                (alt
                 (((goal none)
                   (alt
                    (((benef none))
                     ((benef any)
                      (alt
                       (((benef ((cat pp))) (pattern (dots benef pound)))
                        ((benef ((cat np))) (pattern (dots pp-obj pound)))
                        (pp-obj
                         ((cat pp)
                          (np (^ benef))
                          (prep ((lex to))))))))))))
                  ((goal any)
                   (goal ((alt (((cat np) (gap none))
                                ((cat gap) (gap any) (gap +))))))
                   (alt
                    (((benef none) (pattern (dots goal pound)))
                     ((benef any)
                      (alt
                       (((benef ((cat pp))) (pattern (dots goal benef pound)))
                        ((benef ((cat np)))
                         (alt
                          (((dative +)
                            (pattern (dots benef goal pound)))
```

```
                         ((dative any) (dative -)
                          (pattern (dots goal pp-obj pound))
                          (pp-obj
                           ((cat pp)
                            (np (^ benef))
                            (prep ((lex to)))))))))))))))))
% ...........................................................................
          )
% ---------------------------------------------------------------------------
% VERB-VOICE PASSIVE (DEFAULT OR EXPLICIT)
% NB : WHEN EXPANDING GRAMMAR WITH BENEF,
%      THEN CHANGE PATTERN ...verb by-obj... INTO ...verb dots by-obj...
          ((alt (((verb ((voice passive)))
                  (prot none))
                 ((verb ((voice any) (voice passive)))
                  (prot any)
                  (prot ((alt (((cat np) (gap none))
                               ((cat gap) (gap any) (gap +))))))
                  (by-obj ((cat pp) (prep ((lex by))) (np (^ prot))))
                  (pattern (dots verb dots by-obj dots pound)))
                ))
           (verb ((cat verb-group)))
% ...........................................................................
           (alt
            (((goal none)
              (alt
               (((benef none)
                 (verb (number sing))
                 (pattern (dots verb dots pound)))
                ((benef any) (benef ((cat np)))
                 (verb ((number (^ benef number))))
                 (pattern (benef dots verb dots pound))))))
             ((goal any)
              (goal ((alt (((cat np) (gap none))
                           ((cat gap) (gap any) (gap +))))))
              (benef none)
              (verb ((number (^ goal number))))
              (pattern (goal dots verb dots pound)))
             ((goal any)
              (goal ((alt (((cat np) (gap none))
                           ((cat gap) (gap any) (gap +))))))
              (benef any)
              (alt
               (((benef ((cat pp)))
                 (pattern (goal dots verb benef dots pound))
                 (verb ((number (^ goal number)))))
                ((benef ((cat np)))
                 (alt
                  (((pattern (benef dots verb goal dots pound))
                    (verb ((number (^ benef number)))))
                   ((pattern (goal dots verb pp-obj dots pound))
                    (verb ((number (^ goal number))))
                    (pp-obj
                     ((cat pp)
                      (np (^ benef))
                      (prep ((lex to))))))))))))))
% ...........................................................................
          )
% ---------------------------------------------------------------------------
% VERB VOICE THERE-INSERTION
% I GUESS THERE IS A MISTAKE WITH V AND V1 ? TJOELIONG - 5 July 1985 1415-EDT
         ((verb ((voice any)))
          (verb ((voice there-insertion)))
          (verb ((v ((lex be)))))
          (v1 ((number (^ prot number))))
          (v1 ((tense (^ verb tense))))
```

```
                (pattern (subj v1 dots prot verb dots pound))
                (subj ((cat noun) (lex there)))
                (v1
                 ((cat verb-group)
                  (voice active)
                  (v ((lex (^ ^ verb v lex))))
                  (tense (^ verb tense))
                  (number (^ prot number))))
                (prot ((cat np)))
                (verb
                 ((cat verb-group)
                  (voice there-insertion)
                  (v ((lex any) (cat gap) (gap +)))))
                (alt
                 (((goal none))
                  ((goal any)
                   (goal ((cat np)))
                   (pattern (dots goal pound))))))
% ------------------------------------------------------------------------
              ))
% END VERB VOICE
% ------------------------------------------------------------------------
             (alt
              (((sentence-adverbial none))
               ((sentence-adverbial any)
                (pattern (pound sentence-adverbial dots))
                (sentence-adverbial
                 ((cat s-adv)
                  (punctuation ((before ",") (after ","))))))))
             (alt
              (((sublist none))
               ((sublist any)
                (pattern (dots sublist))
                (sublist
                 ((cat np)
                  (punctuation ((before ":"))))))))
             )
% END SUBORD NONE
%
% --- SUBORDINATE IN ADV-POSITION --- TJOELIONG, BOSTON 17-JULY-1985
            ((subord any) (subord ((cat s-bar)))
             (s ((cat s)))
             (alt
              (
               % SUBORD IN SENTENCE-FINAL POSITION
               ((order post) (pattern (s subord))
                (s ((punctuation ((after ",")))))
               % SUBORD IN SENTENCE-INITIAL POSITION
               ((order any) (order pre) (pattern (subord s))
                (subord ((punctuation ((after ","))))))
              )))
% ------------------------------------------------------------------------
             )))
% END CONJ NONE
% ------------------------------------------------------------------------
        ((conj any) (conj ((cat conj) (lex any)))
         (pattern (slist conj s))
         (slist ((cat slist)))
         (s ((cat s))))
% ------------------------------------------------------------------------
       )))
% 02 CAT SLIST ----------------------------------------------------------
     ((cat slist)
      (alt
       (((s none))
        ((s any)
```

```
          (pattern (slist s))
          (slist ((cat slist)))
          (s ((cat s)))
          (punctuation ((after ","))))))))
% 03 CAT VERB-GROUP - TJOELIONG, 29-APR-1985 ----------------------------
      ((cat verb-group)
       (alt
        (((voice active)
          (pattern (v dots))
          (v ((cat verb) (lex any))))
         ((voice passive)
          (pattern (v1 v dots))
          (v1 ((cat verb) (lex be)))
          (v ((cat verb) (lex any) (tense pastp))))
         ((voice there-insertion)
          (pattern (v dots))
          (v ((lex any) (cat gap) (gap +))))
        ))
       (alt (((pp none)) ((pp any) (pp ((cat pp)))
                          (pattern (dots pp)))))
       (alt (((adv none)) ((adv any) (adv ((cat adv) (lex any)))
                           (pattern (dots adv dots)))))
       )
% 04 CAT NP ------------------------------------------------------------
      ((cat np)
       (alt
        (((embed none)
          (alt
           (((conj none)
             (alt
              (((possessive none)
                (alt
                 (((article none) (pattern (dots nnp dots)))
                  ((article any) (article ((cat article) (lex any)))
                   (pattern (dots article nnp dots)))
                  )))
               ((possessive any)
                (possessive ((cat nnp) (n ((feature possessive)))))
                (alt
                 (((article none) (pattern (possessive nnp dots)))
                  ((article any) (article ((cat article) (lex any)))
                   (pattern (article possessive nnp dots)))
                  )))))
             (nnp ((cat nnp)))
             (alt
              (((parenthetical none))
               ((parenthetical any) (parenthetical ((cat parenthetical)))
                (pattern (dots parenthetical))))))
            ((conj any) (conj ((cat conj) (lex any)))
             (pattern (nplist conj np))
             (nplist ((cat nplist)))
             (np ((cat np))))
            )))
% EMBED = SUBORDINATE IN NP-POSITION - TJOELIONG, 16-MAY-1985
         ((embed any)
          (embed ((cat s-bar)))
          (pattern (dots embed dots)))
        )))
% 05 CAT NPLIST ---------------------------------------------------------
      ((cat nplist)
       (alt
        (((np none))
         ((np any)
          (pattern (nplist np))
          (nplist ((cat nplist)))
          (np ((cat np)))
```

```
        (punctuation ((after ","))))))))
% 06 CAT NNP ------------------------------------------------------------
     ((cat nnp)
      (alt
       (((adj none)
         (pp none)
         (tail none)
         (pattern (n pound))
         (n ((cat noun) (lex any)))
         (alt
          (((sublist none))
           ((sublist any)
            (sublist ((cat np)))
            (pattern (pound sublist))))))
        ((alt
          (((adj any) (adj ((cat adj) (lex any))) (pattern (adj nnp)))
           ((pp any) (pp ((cat pp))) (pattern (nnp pp)))
           ((tail any) (tail ((cat s-bar))) (pattern (nnp tail)))))
         (nnp ((cat nnp))))
        )))
% 07 CAT S-BAR ----------------------------------------------------------
     ((cat s-bar)
      (s ((cat s)))
      (pattern (dots s))
      (alt
       (((relpro none))
        ((relpro any)
         (relpro ((cat relpro) (lex any)))
         (pattern (relpro dots)))))))
% 08 CAT PP -------------------------------------------------------------
     ((cat pp)
      (alt
       (((conj none)
         (pattern (prep np))
         (prep ((cat prep) (lex any)))
         (np ((cat np))))
        ((conj any)
         (conj ((cat conj) (lex any)))
         (pattern (pplist conj pp))
         (pplist ((cat pplist)))
         (pp ((cat pp))))
        )))
% 09 CAT PPLIST ---------------------------------------------------------
     ((cat pplist)
      (alt
       (((pp none))
        ((pp any)
         (pattern (pplist pp))
         (pplist ((cat pplist)))
         (pp ((cat pp)))
         (punctuation ((after ","))))))))
% 10 CAT PARENTHETICAL --------------------------------------------------
     ((cat parenthetical)
      (punctuation ((before "(") (after ")")))
      (alt
       (((sentence-adverbial none) (pattern (dots paren-np)))
        ((sentence-adverbial any) (pattern (dots sentence-adverbial paren-np))
         (sentence-adverbial ((cat s-adv) (punctuation ((after ",")))))))))
      (paren-np ((cat np))))
% 11 CAT GAP ------------------------------------------------------------
     ((cat gap) (gap any) (gap +))
% ------------------------------------------------------------------------
% LEXICON
% ------------------------------------------------------------------------
% LEXICON :
     ((cat article) (alt (((lex indef)) ((lex def)))))
```

```
      ((cat adv) (alt (((lex self-propelled)) ((lex as)))))
      ((cat punctuation) (alt (((lex "(")) ((lex ")")) ((lex ",")))))
% ..................................................................
      ((cat relpro)
       (alt
        (((lex that)) ((lex which)) ((lex who))
         ((lex after))
         ((lex because))
         ((lex before))
         ((lex if))
         ((lex since))
         ((lex when))
         ((lex if)))))
% ..................................................................
      ((cat conj)
       (alt
        (((lex and)) ((lex or))
         ((lex although)) ((lex through)))))
% ..................................................................
      ((cat prep)
       (alt
        (((lex in))
         ((lex on))
         ((lex by))
         ((lex of))
         ((lex with))
         ((lex between))
         ((lex as))
         ((lex prep))
         ((lex to))
         ((lex into))
         ((lex than)))))
% ..................................................................
      ((cat verb)
       (alt
        (((lex attract))
         ((lex cause))
         ((lex compress))
         ((lex decompress))
         ((lex decrease))
         ((lex hit))
         ((lex increase))
         ((lex move))
         ((lex release))
         ((lex speak))
         ((lex vibrate))
         ((lex vary)
          (present (sing (first vary) (second vary) (third varies))
                   (plur (first vary) (second vary) (third vary)))
          (pastp (sing (first varied) (second varied) (third varied))
                 (plur (first varied) (second varied) (third varied))))
         ((lex have)
          (present (sing (first have) (second have) (third has))
                   (plur (first have) (second have) (third have))))
         ((lex be)
          (present (sing (first am) (second are) (third is))
                   (plur (first are) (second are) (third are))))
        )))
% ..................................................................
      ((cat s-adv)
       (alt (((lex forpoundexample)) ((lex therefore)) ((lex also)))))
% ..................................................................
      ((cat adj)
       (alt
        (((lex mean))
         ((lex air))
```

```
          ((lex contact))
          ((lex current))
          ((lex receiver))
          ((lex sound))
          ((lex transmitter))
          ((lex adj))
          ((lex no)))))
% ..........................................................................
      ((cat noun)
       (alt
        (((lex fact))
         ((lex dog))
         ((lex dogs))
         ((lex telephone))
         ((lex housing))
         ((lex line))
         ((lex receiver))
         ((lex cover))
         ((lex kind))
         ((lex diaphragm))
         ((lex granule))
         ((lex intensity))
         ((lex molecule))
         ((lex one))
         ((lex person))
         ((lex pole))
         ((lex resistance))
         ((lex transmitter))
         ((lex wave))
         ((lex WEIGHT_UNITS)))))
% ..........................................................................
     ))))
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
(def list-grammar (lambda nil (pp grammar)))
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
(def lexicon (lambda (grammar)
    %  **)
    %  ** ******************************************************)
    %  **)
    %  ** Returns: the lexicon of the grammar --- The number of cdrs)
    %  ** depends upon the number of categories in the grammar)
    %  **)
    %  ** ******************************************************)
    %  **)
    (Cnth (car (last (car grammar))) 12)))
% ----------------------------------------------------------------------
% ----------------------------------------------------------------------
```

I. A Simple Grammar

```
% The grammars Zgr0, Zgr1, and Zgr2 are meant to be tutorials:
% They show how to increment a grammar to add features.

% TL-ZGR0.LSP - 20 Aug 1985 1107-EDT
% contains : CAT S : active,passive / CAT VERB-GROUP : pp /
%            CAT NMP : adj,pp   ------------------------------

% The grammar is a large functional description: at its top
% level, it contains alternatives: (cat S), (Cat SLIST),
% (cat VERB-GROUP), (cat NP), (cat NMP), (cat NPLIST),
% (cat S-BAR), (cat PP), (cat PPLIST), (cat PARENTHICAL),
% (cat GAP), and finally, the LEXICON.

% In this grammar, SLIST, NPLIST, S-BAR, PPLIST, PARENTHETICAL,
% and GAP have been omitted for simplicity.

% Note the attribute value pairs which describe the grammar,
% and the patterns that show the order in which the constituent
% should appear in the final sentence.

% This grammar allows for default passive (when no voice is
% mentioned in the input but there is no protagonist).
```

```
(setq grammar
 '((alt
    (
% 01 CAT S : sentence ------------------------------------
     ((cat s)
      (alt
       (
% VERB VOICE ACTIVE (DEFAULT OR EXPLICIT) ---------------------
% There MUST be a protagonist

        ((verb ((voice active)))
         (prot any)
         (prot ((cat np)))
         (pattern (prot dots verb dots pound))
         (verb
          ((cat verb-group)
           (number (↑ prot number))))
% Is there a goal? ..........................................
        (alt
         (
          ((goal none))
          ((goal any)
           (goal ((cat np)))
           (pattern (dots goal pound)))))
% ..........................................................
        )

% VERB-VOICE PASSIVE (DEFAULT OR EXPLICIT)  ------------------
% There may not be a protagonist.

        ((alt (((verb ((voice passive)))
                 (prot none))
                ((verb ((voice any) (voice passive)))
                 (prot any)
                 (prot ((cat np)))
                 (by-obj ((cat pp) (prep ((lex by)))
                                   (np (↑ prot))))
                 (pattern (dots verb by-obj dots pound)))))
         (verb ((cat verb-group)))
% ..........................................................
        (alt
         (
          ((goal none)
           (pattern (dots verb dots))
           (verb (number sing)))
          ((goal any)
           (goal ((cat np)))
           (pattern (goal dots verb dots pound))
           (verb ((number (↑ goal number))))))))
% ..........................................................
        )
% ----------------------------------------------------------
       )))
% 02 CAT SLIST : for coordinated main clauses-----------------
     ((cat slist))

% 03 CAT VERB-GROUP - TJOELIONG, 29-APR-1985 ----------------
     ((cat verb-group)
      (alt
       (
        ((voice active)
         (pattern (v dots))
         (v ((cat verb) (lex any))))
```

```
        ((voice passive)
         (pattern (v1 v dots))
         (v1 ((cat verb) (lex be)))
         (v ((cat verb) (lex any) (tense pastp))))))
       (alt
        (
         ((pp none))
         ((pp any)
          (pp ((cat pp)))
          (pattern (dots pp)))))
       )

% 04 CAT NP : for noun phrase ---------------------------------
     ((cat np)
      (alt
       (
        ((article none)
         (pattern (dots nnp dots)))
        ((article any)
         (article ((cat article) (lex any)))
         (pattern (dots article nnp dots)))))
      (nnp ((cat nnp))))
% 05 CAT NPLIST : for coordinated noun phrases ----------------
     ((cat nplist))

% 06 CAT NNP : Noun phrases, WITHOUT article ------------------
     ((cat nnp)
      (alt
       (
        ((adj none)
         (pp none)
         (tail none)
         (pattern (n pound))
         (n ((cat noun) (lex any))))
        ((alt
          (
           ((adj any)
            (adj ((cat adj) (lex any)))
            (pattern (adj nnp)))
           ((pp any)
            (pp ((cat pp)))
            (pattern (nnp pp)))))
         (nnp ((cat nnp))))
       )))

% 07 CAT S-BAR : for subordinate clauses ----------------------
     ((cat s-bar))

% 08 CAT PP : for prepositional phrases -----------------------
     ((cat pp)
      (pattern (prep np))
      (prep ((cat prep) (lex any)))
      (np ((cat np)))
      )

% 09 CAT PPLIST : for coordinated prepositional phrases -------
     ((cat pplist))
% 10 CAT PARENTHETICAL ----------------------------------------
     ((cat parenthetical))
% 11 CAT GAP --------------------------------------------------
     ((cat gap))

% ------------------------------------------------------------
% LEXICON :
     ((cat article) (alt (((lex indef)) ((lex def)))))
```

```
% ..........................................................
     ((cat prep)
      (alt
       (((lex in))
        ((lex into))
        ((lex of))
        ((lex on))
        ((lex to))
       )))
% ..........................................................
     ((cat verb)
      (alt
       (((lex attract))
        ((lex cause))
        ((lex compress))
        ((lex hit))
        ((lex increase))
        ((lex move))
        ((lex speak))
        ((lex vibrate))
        ((lex vary)
         (present (sing (first vary) (second vary)
                        (third varies))
                  (plur (first vary) (second vary)
                        (third vary)))
         (pastp (sing (first varied) (second varied)
                      (third varied))
                (plur (first varied) (second varied)
                      (third varied))))
        ((lex have)
         (present (sing (first have) (second have)
                        (third has))
                  (plur (first have) (second have)
                        (third have))))
        ((lex be)
         (present (sing (first am) (second are) (third is))
                  (plur (first are) (second are) (third are))))
       )))
% ..........................................................
     ((cat adj)
      (alt
       (((lex big))
        ((lex current))
        ((lex new))
        ((lex old))
        ((lex small))
        ((lex sound))
       )))
% ..........................................................
     ((cat noun)
      (alt
       (((lex telephone))
        ((lex receiver))
        ((lex kind))
        ((lex diaphragm))
        ((lex granule))
        ((lex intensity))
        ((lex molecule))
        ((lex person))
        ((lex pole))
        ((lex resistance))
        ((lex transmitter))
        ((lex wave))
       )))
% ----------------------------------------------------------
```

```
    ))))
% ------------------------------------------------------------
(def list-grammar (lambda nil (pp grammar)))
% ------------------------------------------------------------
(def lexicon (lambda (grammar)
    %  **
    %  ** Returns: the lexicon of the grammar --- The number of
    %  ** cdrs depends upon the number of categories in the
    %  ** grammar
    (Cnth (car (last (car grammar))) 12)))
% ------------------------------------------------------------
```

II. Using the Unifier

The unifier can be used in PSL (both on the dec-20 and on the hp). An NMODE INIT file is needed for PSL initialization.

On the dec-20, to use the unifier, you need to load the following files:

1. <kwee>tl-work.lsp: this file loads all the necessary files, including the new unifier (**<kwee>c-ucon.lsp**) and the file containing the changes to the linearizer (**<kwee>tl-linear-4plus.lsp**)[19]

2. A file containing the grammar you want to use. The directory **<kwee>** contains sample grammars and the final complete one, **tl-fullgram-8**[20]

3. The file containing the inputs. The directory **<kwee>** contains sample inputs too.

---

[19] Depending on your init file, you may be asked whether you really want to redefine a system function. Type **yes**.

[20] Note that this grammar is the complete grammar taken from [McKeown 82]. The parts tested and changed by TjoeLiong Kwee are incorporated in it. There are however parts that have not yet been tested. See TjoeLiong's report for more information.

III. Glossary of symbols in the unifier:

Variables

ALT: alternative
AV: attribute value pair
CSET: constituent set
FD: Functional Description
G: Grammar
GVAL: Grammar value (value from the attribute-value pair from the grammar)
I: Input
IVAL: Input value (value from the attribute-value pair from the input)
LS: Loose
PATH: a path corresponds to the attribute in the pair found in the grammar.

-RES: result from...
UCONFNS: Ucon functions: Functions used in the unifier

Functions

ALT-H-LS: alternative handling with LOOSE
AV-H-LS: attribute value handling with LOOSE
CNTRL-AV-H-LS: central attribute value handling with Loose
CSET-H-LS: Constituent set handling and Loose
UCON: unify constituents
UCON-MANY: Unify many constituents
UAVS-LS: unify attribute value pairs and Loose

IV. Inventory of the files left by TjoeLiong

- Tl-work.lsp: This file initializes the unifier: loads the appropriate files

- C-ucon.lsp: unifier containing the changes made by Cecile (also contain the change that had been done ny Michal).

- C-ucon-chg.lsp: contains only the functions that have been changed by Cecile

- C-ucon-comm.txt: contains comments on Cecile's changes of the unifier

- Michal-ucon.lsp: Unifier with the change done by Michal.

- Tl-Fullgram-8.lsp: this is the current full grammar. Parts of it has been fully tested (see TjoeLiong's documentation). The other parts are copied from Kathy's grammar.

- Zgr0.lsp, Zgr1.lsp, Zgr2.lsp : These are intended as a tutorial to show how one can increment a grammar. Each of these grammars adds one feature to the original grammar. [These features are incorporated in the full actual grammar].

- Zgr3.lsp through Zgr9.lsp: These grammars are test-grammars: variants of zgr3 and zgr4 to test ideas.

    * Zgr3.lsp: This grammar is Zgr0 with the cat NNP, TAIL and GAP. Here, GAP and NP are different categories. Zgr3-test-adj-pp-tail.lsp is a file containing tests for this grammar. This file treats the GAP as the original grammar (Kathy's grammar) did.

    * Zgr4.lsp: This grammar tests the POSSESSIVE. Here, POSSESSIVE is in cat NP. The file Zgr4-test-poss1.lsp is a file containing tests for this grammar. This file treat the possessive as the original grammar did.

    * Zgr5.lsp: In this grammar, POSSESSIVE is in cat NNP.

    * Zgr6.lsp: In this grammar, POSSESSIVE is in cat NP, but ARTICLE and NP are mutually exclusive.

    * Zgr7.lsp: This grammar tests GAP. Here, GAP is an alternative inside cat NP. The file Zgr7-test-5 contain inputs for this grammar.

    * Zgr8.lsp: This grammar tests GAP. GAP and NP are under 1

category N-BAR. This category appears only where a GAP is allowed. The file Zgr8-test-5 contain inputs for this grammar.

* Zgr9.lsp: This grammar tests GAP. In the CAT PP category, an alternative is added to allow for a GAP. The file Zgr9-test-5 contain inputs for this grammar.

- Tl-DocXX.lsp: these contain old documentation and grammars written by TjoeLiong.

- Tl-linear-4plus.lsp: Changes made to the unifier by TjoeLiong and Cecile.

- Tl-report-XX.lsp: These files make up the report written by TjoeLiong. The report is about the changes and additions TjoeLiong made to Kathy's grammar.

- Tl-test-XX: these files contain test inputs (and outputs) for the changes TjoeLiong have done.

- Ztest-poss1.lsp: tests POSSESSIVE.

- Ztest-adj-pp-tail.lsp tests TAIL.

V. Illustration of the second remaining problem mentioned

```
(SETQ X
      '((CAT NP) (ARTICLE === DEF) (N === TRANSMITTER) (NUMBER PLUR)
        (PP ((PREP === IN) (N === TELEPHONE) (ARTICLE === DEF)))))

(THE TRANSMITTERS IN THE TELEPHONE!.)

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

(SETQ X
      '((CAT NP) (ARTICLE === DEF) (N === TRANSMITTER) (NUMBER PLUR)
        (PP ((PREP === IN) (N === TELEPHONE) (ARTICLE === INDEF)))))

(THE TRANSMITTERS IN TELEPHONE!.)

% Note that the (ARTICLE === INDEF) attached to TELEPHONE disappeared
% This only happens when TRANSMITTER is plural and the article is
% indefinite

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

(SETQ X
      '((CAT NP) (ARTICLE === DEF) (N === TRANSMITTER)
        (PP ((PREP === IN) (N === TELEPHONE) (ARTICLE === DEF)))))

(THE TRANSMITTER IN THE TELEPHONE!.)

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

(SETQ X
      '((CAT NP) (ARTICLE === DEF) (N === TRANSMITTER)
        (PP ((PREP === IN) (N === TELEPHONE) (ARTICLE === INDEF)))))

(THE TRANSMITTER IN A TELEPHONE!.)
```

```
% Trace to show what the output of the unification (before
% linearization) is:

(SETQ X
      '((CAT NP) (ARTICLE === DEF) (N === TRANSMITTER) (NUMBER PLUR)
        (PP ((PREP === IN) (N === TELEPHONE) (ARTICLE === INDEF)))))

% After unifying (CAT NP), the top-level constituent:

(SETQ UAVSRES
      '(((NNP ((CAT NNP))) (PATTERN (DOTS ARTICLE NNP DOTS))
        (ARTICLE ((LEX DEF) (CAT ARTICLE))) (CAT NP)
        (N ((LEX TRANSMITTER))) (NUMBER PLUR)
        (PP ((PREP ((LEX IN))) (N ((LEX TELEPHONE)))
             (ARTICLE ((LEX INDEF))))))
       NIL))

% Before linearization:

(SETQ UCONRES
 '((((NNP ((PP ((NP ((NNP ((N ((LEX TELEPHONE) (CAT NOUN))) (CSET (N))
                            (PATTERN (N POUND)) (TAIL NONE) (PP NONE)
                            (ADJ NONE) (CAT NNP)))
                      (ARTICLE ((LEX INDEF) (CAT ARTICLE)))
                      (CSET (ARTICLE NNP))
                      (PATTERN (DOTS ARTICLE NNP DOTS)) (CAT NP)))
                (PREP ((LEX IN) (CAT PREP))) (CSET (PREP NP))
                (PATTERN (PREP NP)) (CAT PP)))
           (NNP ((N ((LEX TRANSMITTER) (CAT NOUN))) (CSET (N))
                 (PATTERN (N POUND)) (TAIL NONE) (PP NONE) (ADJ NONE)
                 (CAT NNP)))
           (CSET (NNP PP)) (PATTERN (NNP PP)) (CAT NNP)))
     (ARTICLE ((LEX DEF) (CAT ARTICLE))) (CSET (ARTICLE NNP))
     (PATTERN (DOTS ARTICLE NNP DOTS)) (CAT NP) (NUMBER PLUR))
    ((TAIL NONE) (PP NONE) (ADJ NONE) (NUMBER PLUR)))))

(THE TRANSMITTERS IN TELEPHONE! )

% Note that, at this point, the article is still attached to
% TELEPHONE, and (NUMBER PLURAL) refers to the top-level constituent,
% TRANSMITTER.
```

VI. Trace of the unification of a simple input with the grammar shown in Appendix I

Trace of the unification of a simple input with the grammar shown in appendix I

```
% Tracing of a simple imput withe the grammar given in the appendix.
% The input is fully specified, so that there is no use for LOOSE.
% I edited this file and took whatever dealt with LOOSE out.

% with tracing on: inside-tracing

(SETQ X2
      '((CAT NP) (ARTICLE === DEF)
        (NNP ((ADJ === SMALL)
              (NNP ((ADJ === NEW)
                    (NNP ((ADJ === LIGHT)
                          (NNP ((N === GRANULE)))))))))))

** In uavs-ls:

(SETQ I
 '((CAT NP) (ARTICLE ((LEX DEF)))
   (NNP ((ADJ ((LEX SMALL)))
         (NNP ((ADJ ((LEX NEW)))
               (NNP ((ADJ ((LEX LIGHT)))
                     (NNP ((N ((LEX GRANULE)))))))))))

 G == the whole grammar
(SETQ GA 'ALT)


The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I
 '((CAT NP) (ARTICLE ((LEX DEF)))
   (NNP ((ADJ ((LEX SMALL)))
         (NNP ((ADJ ((LEX NEW)))
               (NNP ((ADJ ((LEX LIGHT)))
                     (NNP ((N ((LEX GRANULE)))))))))))

(SETQ G
      '((ALT (((ARTICLE NONE) (PATTERN (DOTS NNP DOTS)))
              ((ARTICLE ANY) (ARTICLE ((CAT ARTICLE) (LEX ANY)))
               (PATTERN (DOTS ARTICLE NNP DOTS)))))
        (NNP ((CAT NNP)))))


(SETQ GA 'ALT)


** In uavs-ls:

(SETQ I
 '((CAT NP) (ARTICLE ((LEX DEF)))
   (NNP ((ADJ ((LEX SMALL)))
         (NNP ((ADJ ((LEX NEW)))
               (NNP ((ADJ ((LEX LIGHT)))
                     (NNP ((N ((LEX GRANULE)))))))))))

(SETQ G '((ARTICLE NONE) (PATTERN (DOTS NNP DOTS)) (NNP ((CAT NNP)))))

(SETQ GA 'ARTICLE)

 Calling av-h-ls
```

```
 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT ARTICLE))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same  PATH was found in the input

(SETQ IVAL 'NIL)


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'ARTICLE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((ARTICLE NIL)))

(SETQ PATH 'CAT)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT ARTICLE) (LEX DEF)))

(SETQ G '((LEX ANY)))


(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX ANY))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'DEF)


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:
```

```
(SETQ I 'DEF)

(SETQ G 'ANY)

Out of uavs-ls

Success: will enrich the input with
(SETQ RESULTS '((DEF NIL)))

(SETQ PATH 'LEX)

returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ G 'NIL)

Out of uavs-ls

Success: will enrich the input with
(SETQ RESULTS '((((LEX DEF) (CAT ARTICLE)) NIL)))

(SETQ PATH 'ARTICLE)

returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
 '((ARTICLE ((LEX DEF) (CAT ARTICLE))) (CAT NP)
   (NNP ((ADJ ((LEX SMALL)))
         (NNP ((ADJ ((LEX NEW)))
               (NNP ((ADJ ((LEX LIGHT)))
                     (NNP ((N ((LEX GRANULE)))))))))))))

(SETQ G '((PATTERN (DOTS ARTICLE NNP DOTS)) (NNP ((CAT NNP)))))

(SETQ GA 'PATTERN)

** In uavs-ls:

(SETQ I
 '((PATTERN (DOTS ARTICLE NNP DOTS))
   (ARTICLE ((LEX DEF) (CAT ARTICLE)))
   (CAT NP)
   (NNP ((ADJ ((LEX SMALL)))
         (NNP ((ADJ ((LEX NEW)))
               (NNP ((ADJ ((LEX LIGHT)))
                     (NNP ((N ((LEX GRANULE)))))))))))))
(SETQ G '((NNP ((CAT NNP)))))

(SETQ GA 'NNP)

Calling av-h-ls
*** In av-h-ls
```

```
******* Attribute-value pair from the grammar:
(SETQ GAV '(NNP ((CAT NNP))))

*** Calling cntrl-av-h-ls to unify the
input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL
      '((ADJ ((LEX SMALL)))
        (NNP ((ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE))))))))))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I
      '((ADJ ((LEX SMALL)))
        (NNP ((ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE))))))))))

(SETQ G '((CAT NNP)))

(SETQ GA 'CAT)

Calling av-h-ls
*** In av-h-ls

******* Attribute-value pair from the grammar:
(SETQ GAV '(CAT NNP))

*** Calling cntrl-av-h-ls to unify the
input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same  PATH was found in the input

(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'NNP)

Out of uavs-ls

Success: will enrich the input with
(SETQ RESULTS '((NNP NIL)))

(SETQ PATH 'CAT)
```

```
 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((CAT NNP) (ADJ ((LEX SMALL)))
        (NNP ((ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE)))))))))))

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS
 '((((CAT NNP) (ADJ ((LEX SMALL)))
     (NNP ((ADJ ((LEX NEW)))
           (NNP ((ADJ ((LEX LIGHT)))
                 (NNP ((N ((LEX GRANULE)))))))))
    NIL)))

(SETQ PATH 'NNP)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
 '((NNP ((CAT NNP) (ADJ ((LEX SMALL)))
         (NNP ((ADJ ((LEX NEW)))
               (NNP ((ADJ ((LEX LIGHT)))
                     (NNP ((N ((LEX GRANULE)))))))))
   (PATTERN (DOTS ARTICLE NNP DOTS))
   (ARTICLE ((LEX DEF) (CAT ARTICLE)))
   (CAT NP)))


(SETQ G 'NIL)


Alternative succeeded
(SETQ RESULT-0
 '((((NNP ((CAT NNP) (ADJ ((LEX SMALL)))
           (NNP ((ADJ ((LEX NEW)))
                 (NNP ((ADJ ((LEX LIGHT)))
                       (NNP ((N ((LEX GRANULE)))))))))
     (PATTERN (DOTS ARTICLE NNP DOTS))
     (ARTICLE ((LEX DEF) (CAT ARTICLE)))
     (CAT NP))
    NIL)))


about to quit alt-h-ls
** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

 G == the whole grammar
(SETQ GA 'ALT)
```

```
The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ G '((ALT (((LEX INDEF)) ((LEX DEF))))))

(SETQ GA 'ALT)

lexical level

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ I-LEX '(LEX DEF))

(SETQ G-LEX '((LEX DEF)))

** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ G '((LEX DEF)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX DEF))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'DEF)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'DEF)

(SETQ G 'DEF)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((DEF NIL)))

(SETQ PATH 'LEX)

returning the result of a call to uavs-many-ls
```

```
** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ G 'NIL)


** In uavs-ls:

(SETQ I
      '((CAT NNP) (ADJ ((LEX SMALL)))
        (NNP ((ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE)))))))))))

 G == the whole grammar
(SETQ GA 'ALT)


The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I
      '((CAT NNP) (ADJ ((LEX SMALL)))
        (NNP ((ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE)))))))))))

(SETQ G
      '((ALT (((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
               (N ((CAT NOUN) (LEX ANY))))
              ((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY)))
                      (PATTERN (ADJ NNP)))
                     ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
               (NNP ((CAT NNP)))))))))


(SETQ GA 'ALT)


** In uavs-ls:

(SETQ I
      '((CAT NNP) (ADJ ((LEX SMALL)))
        (NNP ((ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE)))))))))))

(SETQ G
      '((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
        (N ((CAT NOUN) (LEX ANY)))))


(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ NONE))


 *** Calling cntrl-av-h-ls to unify the
```

```
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX SMALL)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX SMALL)))

(SETQ G 'NONE)

 Out of uavs-ls
Unifying the 2 values failed

 returning the result of a call to uavs-many-ls

 Alternative has failed...
** In uavs-ls:

(SETQ I
      '((CAT NNP) (ADJ ((LEX SMALL)))
        (NNP ((ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE)))))))))))

(SETQ G
      '((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY)))
                         (PATTERN (ADJ NNP)))
              ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
        (NNP ((CAT NNP)))))


(SETQ GA 'ALT)


** In uavs-ls:

(SETQ I
      '((CAT NNP) (ADJ ((LEX SMALL)))
        (NNP ((ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE)))))))))))

(SETQ G
      '((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY))) (PATTERN (ADJ NNP))
        (NNP ((CAT NNP)))))


(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ ANY))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar
```

```
In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX SMALL)))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX SMALL)))

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX SMALL)) NIL)))

(SETQ PATH 'ADJ)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((ADJ ((LEX SMALL))) (CAT NNP)
        (NNP ((ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE)))))))))))

(SETQ G
      '((ADJ ((CAT ADJ) (LEX ANY))) (PATTERN (ADJ NNP))
                                    (NNP ((CAT NNP)))))


(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ ((CAT ADJ) (LEX ANY))))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX SMALL)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX SMALL)))

(SETQ G '((CAT ADJ) (LEX ANY)))
```

```
(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT ADJ))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same  PATH was found in the input

(SETQ IVAL 'NIL)


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'ADJ)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((ADJ NIL)))

(SETQ PATH 'CAT)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT ADJ) (LEX SMALL)))

(SETQ G '((LEX ANY)))


(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX ANY))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'SMALL)


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
```

```
** In uavs-ls:

(SETQ I 'SMALL)

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((SMALL NIL)))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX SMALL) (CAT ADJ)))

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX SMALL) (CAT ADJ)) NIL)))

(SETQ PATH 'ADJ)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((ADJ ((LEX SMALL) (CAT ADJ))) (CAT NNP)
        (NNP ((ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE)))))))))))

(SETQ G '((PATTERN (ADJ NNP)) (NNP ((CAT NNP)))))

(SETQ GA 'PATTERN)

** In uavs-ls:

(SETQ I
      '((PATTERN (ADJ NNP)) (ADJ ((LEX SMALL) (CAT ADJ)))
        (CAT NNP)
        (NNP ((ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE)))))))))))

(SETQ G '((NNP ((CAT NNP)))))

(SETQ GA 'NNP)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:
```

```
(SETQ GAV '(NNP ((CAT NNP))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL
      '((ADJ ((LEX NEW)))
        (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I
      '((ADJ ((LEX NEW)))
        (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))))

(SETQ G '((CAT NNP)))

(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT NNP))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same  PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'NNP)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NNP NIL)))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:
```

```
(SETQ I
      '((CAT NNP) (ADJ ((LEX NEW)))
        (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))))

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS
      '((((CAT NNP) (ADJ ((LEX NEW)))
          (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))
         NIL)))

(SETQ PATH 'NNP)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((NNP ((CAT NNP) (ADJ ((LEX NEW)))
              (NNP ((ADJ ((LEX LIGHT)))
                    (NNP ((N ((LEX GRANULE)))))))))
        (PATTERN (ADJ NNP)) (ADJ ((LEX SMALL) (CAT ADJ)))
        (CAT NNP)))

(SETQ G 'NIL)


Alternative succeeded
(SETQ RESULT-0
 '((((NNP ((CAT NNP) (ADJ ((LEX NEW)))
           (NNP ((ADJ ((LEX LIGHT)))
                 (NNP ((N ((LEX GRANULE)))))))))
     (PATTERN (ADJ NNP)) (ADJ ((LEX SMALL) (CAT ADJ)))
     (CAT NNP))
    NIL)))


about to quit alt-h-ls
Alternative succeeded
(SETQ RESULT-0
 '((((NNP ((CAT NNP) (ADJ ((LEX NEW)))
           (NNP ((ADJ ((LEX LIGHT)))
                 (NNP ((N ((LEX GRANULE)))))))))
     (PATTERN (ADJ NNP)) (ADJ ((LEX SMALL) (CAT ADJ)))
     (CAT NNP))
    NIL)))


about to quit alt-h-ls
** In uavs-ls:

(SETQ I '((LEX SMALL) (CAT ADJ)))

 G == the whole grammar
(SETQ GA 'ALT)


The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:
```

```
(SETQ I '((LEX SMALL) (CAT ADJ)))

(SETQ G
      '((ALT (((LEX BIG)) ((LEX LIGHT)) ((LEX CURRENT)) ((LEX NEW))
              ((LEX OLD)) ((LEX SMALL)) ((LEX SOUND))))))

(SETQ GA 'ALT)

lexical level

(SETQ I '((LEX SMALL) (CAT ADJ)))

(SETQ I-LEX '(LEX SMALL))

(SETQ G-LEX '((LEX SMALL)))

** In uavs-ls:

(SETQ I '((LEX SMALL) (CAT ADJ)))

(SETQ G '((LEX SMALL)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX SMALL))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'SMALL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'SMALL)

(SETQ G 'SMALL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((SMALL NIL)))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:
```

```
(SETQ I '((LEX SMALL) (CAT ADJ)))

(SETQ G 'NIL)


** In uavs-ls:

(SETQ I
      '((CAT NNP) (ADJ ((LEX NEW)))
        (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))))

 G == the whole grammar
(SETQ GA 'ALT)


The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I
      '((CAT NNP) (ADJ ((LEX NEW)))
        (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))))

(SETQ G
      '((ALT (((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
               (N ((CAT NOUN) (LEX ANY))))
              ((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY)))
                      (PATTERN (ADJ NNP)))
                     ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
               (NNP ((CAT NNP)))))))))


(SETQ GA 'ALT)


** In uavs-ls:

(SETQ I
      '((CAT NNP) (ADJ ((LEX NEW)))
        (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))))

(SETQ G
      '((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
        (N ((CAT NOUN) (LEX ANY)))))


(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ NONE))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX NEW)))
```

```
** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX NEW)))

(SETQ G 'NONE)

 Out of uavs-ls
Unifying the 2 values failed

 returning the result of a call to uavs-many-ls

 Alternative has failed...
** In uavs-ls:

(SETQ I
      '((CAT NNP) (ADJ ((LEX NEW)))
        (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))))

(SETQ G
      '((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY)))
                         (PATTERN (ADJ NNP)))
               ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
        (NNP ((CAT NNP)))))


(SETQ GA 'ALT)


** In uavs-ls:

(SETQ I
      '((CAT NNP) (ADJ ((LEX NEW)))
        (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))))

(SETQ G
      '((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY))) (PATTERN (ADJ NNP))
        (NNP ((CAT NNP)))))


(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ ANY))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX NEW)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX NEW)))
```

```
(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX NEW)) NIL)))

(SETQ PATH 'ADJ)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((ADJ ((LEX NEW))) (CAT NNP)
        (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))))

(SETQ G
      '((ADJ ((CAT ADJ) (LEX ANY))) (PATTERN (ADJ NNP))
                                    (NNP ((CAT NNP)))))


(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ ((CAT ADJ) (LEX ANY))))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX NEW)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX NEW)))

(SETQ G '((CAT ADJ) (LEX ANY)))


(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT ADJ))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar
```

```
In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same  PATH was found in the input

(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'ADJ)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((ADJ NIL)))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT ADJ) (LEX NEW)))

(SETQ G '((LEX ANY)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX ANY))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'NEW)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NEW)

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NEW NIL)))

(SETQ PATH 'LEX)
```

```
 returning the result of a cal' to uavs-many-ls
** In uavs-ls:
(SETQ I '((LEX NEW) (CAT ADJ)))
(SETQ G 'NIL)
 Out of uavs-ls
 Success: will enrich the input with
(SETQ RESULTS '((((LEX NEW) (CAT ADJ)) NIL)))
(SETQ PATH 'ADJ)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
(SETQ I
      '((ADJ ((LEX NEW) (CAT ADJ))) (CAT NNP)
        (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))))
(SETQ G '((PATTERN (ADJ NNP)) (NNP ((CAT NNP)))))

(SETQ GA 'PATTERN)

** In uavs-ls:
(SETQ I
      '((PATTERN (ADJ NNP)) (ADJ ((LEX NEW) (CAT ADJ))) (CAT NNP)
        (NNP ((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))))
(SETQ G '((NNP ((CAT NNP)))))

(SETQ GA 'NNP)
 Calling av-h-ls
 *** In av-h-ls
 ******* Attribute-value pair from the grammar:
(SETQ GAV '(NNP ((CAT NNP))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar
In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
** In uavs-ls:
(SETQ I '((ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))
```

```
(SETQ G '((CAT NNP)))

(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT NNP))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input

(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'NNP)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NNP NIL)))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT NNP) (ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS
      '((((CAT NNP)
          (ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))) NIL)))

(SETQ PATH 'NNP)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((NNP ((CAT NNP) (ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))
        (PATTERN (ADJ NNP)) (ADJ ((LEX NEW) (CAT ADJ))) (CAT NNP)))

(SETQ G 'NIL)
```

```
Alternative succeeded
(SETQ RESULT-0
      '((((NNP ((CAT NNP) (ADJ ((LEX LIGHT)))
                          (NNP ((N ((LEX GRANULE))))))))
         (PATTERN (ADJ NNP)) (ADJ ((LEX NEW) (CAT ADJ))) (CAT NNP))
        NIL)))

about to quit alt-h-ls
Alternative succeeded
(SETQ RESULT-0
      '((((NNP ((CAT NNP) (ADJ ((LEX LIGHT)))
                          (NNP ((N ((LEX GRANULE))))))))
         (PATTERN (ADJ NNP)) (ADJ ((LEX NEW) (CAT ADJ))) (CAT NNP))
        NIL)))

about to quit alt-h-ls
** In uavs-ls:

(SETQ I '((LEX NEW) (CAT ADJ)))

 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((LEX NEW) (CAT ADJ)))

(SETQ G
      '((ALT (((LEX BIG)) ((LEX LIGHT)) ((LEX CURRENT)) ((LEX NEW))
              ((LEX OLD)) ((LEX SMALL)) ((LEX SOUND))))))

(SETQ GA 'ALT)

lexical level

(SETQ I '((LEX NEW) (CAT ADJ)))

(SETQ I-LEX '(LEX NEW))

(SETQ G-LEX '((LEX NEW)))

** In uavs-ls:

(SETQ I '((LEX NEW) (CAT ADJ)))

(SETQ G '((LEX NEW)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:
```

```
(SETQ GAV '(LEX NEW))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'NEW)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NEW)

(SETQ G 'NEW)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NEW NIL)))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX NEW) (CAT ADJ)))

(SETQ G 'NIL)

** In uavs-ls:

(SETQ I '((CAT NNP) (ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE))))))

 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((CAT NNP) (ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE))))))

(SETQ G
      '((ALT (((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
               (N ((CAT NOUN) (LEX ANY))))
              ((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY)))
                      (PATTERN (ADJ NNP)))
                     ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
               (NNP ((CAT NNP)))))))))

(SETQ GA 'ALT)

** In uavs-ls:

(SETQ I '((CAT NNP) (ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE))))))
```

```
(SETQ G
      '((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
        (N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ NONE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX LIGHT)))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX LIGHT)))

(SETQ G 'NONE)

 Out of uavs-ls
Unifying the 2 values failed

 returning the result of a call to uavs-many-ls

 Alternative has failed...
** In uavs-ls:

(SETQ I '((CAT NNP) (ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))

(SETQ G
      '((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY)))
                         (PATTERN (ADJ NNP)))
              ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
        (NNP ((CAT NNP)))))

(SETQ GA 'ALT)

** In uavs-ls:

(SETQ I '((CAT NNP) (ADJ ((LEX LIGHT))) (NNP ((N ((LEX GRANULE)))))))

(SETQ G
      '((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY))) (PATTERN (ADJ NNP))
        (NNP ((CAT NNP)))))

(SETQ GA 'ADJ)

 Calling av-h-ls
```

```
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ ANY))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX LIGHT)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX LIGHT)))

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX LIGHT)) NIL)))

(SETQ PATH 'ADJ)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((ADJ ((LEX LIGHT))) (CAT NNP) (NNP ((N ((LEX GRANULE)))))))

(SETQ G
      '((ADJ ((CAT ADJ) (LEX ANY))) (PATTERN (ADJ NNP))
                                    (NNP ((CAT NNP)))))


(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ ((CAT ADJ) (LEX ANY))))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX LIGHT)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:
```

```
(SETQ I '((LEX LIGHT)))

(SETQ G '((CAT ADJ) (LEX ANY)))

(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT ADJ))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input

(SETQ IVAL 'NIL)


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'ADJ)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((ADJ NIL)))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT ADJ) (LEX LIGHT)))

(SETQ G '((LEX ANY)))


(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX ANY))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'LIGHT)
```

```
** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'LIGHT)

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((LIGHT NIL)))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX LIGHT) (CAT ADJ)))

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX LIGHT) (CAT ADJ)) NIL)))

(SETQ PATH 'ADJ)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((ADJ ((LEX LIGHT) (CAT ADJ))) (CAT NNP)
        (NNP ((N ((LEX GRANULE))))))

(SETQ G '((PATTERN (ADJ NNP)) (NNP ((CAT NNP)))))

(SETQ GA 'PATTERN)

** In uavs-ls:

(SETQ I
      '((PATTERN (ADJ NNP)) (ADJ ((LEX LIGHT) (CAT ADJ))) (CAT NNP)
        (NNP ((N ((LEX GRANULE))))))

(SETQ G '((NNP ((CAT NNP)))))

(SETQ GA 'NNP)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(NNP ((CAT NNP))))
```

```
*** Calling cntrl-av-h-ls to unify the
input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((N ((LEX GRANULE)))))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((N ((LEX GRANULE)))))

(SETQ G '((CAT NNP)))

(SETQ GA 'CAT)

Calling av-h-ls
*** In av-h-ls

******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT NNP))

*** Calling cntrl-av-h-ls to unify the
input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input

(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'NNP)

Out of uavs-ls

Success: will enrich the input with
(SETQ RESULTS '((NNP NIL)))

(SETQ PATH 'CAT)

returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT NNP) (N ((LEX GRANULE)))))

(SETQ G 'NIL)

Out of uavs-ls

Success: will enrich the input with
```

```
(SETQ RESULTS '((((CAT NNP) (N ((LEX GRANULE)))) NIL)))

(SETQ PATH 'NNP)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((NNP ((CAT NNP) (N ((LEX GRANULE))))) (PATTERN (ADJ NNP))
        (ADJ ((LEX LIGHT) (CAT ADJ))) (CAT NNP)))

(SETQ G 'NIL)


Alternative succeeded
(SETQ RESULT-0
      '((((NNP ((CAT NNP) (N ((LEX GRANULE))))) (PATTERN (ADJ NNP))
          (ADJ ((LEX LIGHT) (CAT ADJ))) (CAT NNP))
        NIL)))


about to quit alt-h-ls
Alternative succeeded
(SETQ RESULT-0
      '((((NNP ((CAT NNP) (N ((LEX GRANULE))))) (PATTERN (ADJ NNP))
          (ADJ ((LEX LIGHT) (CAT ADJ))) (CAT NNP))
        NIL)))


about to quit alt-h-ls
** In uavs-ls:

(SETQ I '((LEX LIGHT) (CAT ADJ)))

 G == the whole grammar
(SETQ GA 'ALT)


The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((LEX LIGHT) (CAT ADJ)))

(SETQ G
      '((ALT (((LEX BIG)) ((LEX LIGHT)) ((LEX CURRENT)) ((LEX NEW))
              ((LEX OLD)) ((LEX SMALL)) ((LEX SOUND))))))


(SETQ GA 'ALT)


lexical level

(SETQ I-'((LEX LIGHT) (CAT ADJ)))

(SETQ I-LEX '(LEX LIGHT))

(SETQ G-LEX '((LEX LIGHT)))


** In uavs-ls:
```

```
(SETQ I '((LEX LIGHT) (CAT ADJ)))

(SETQ G '((LEX LIGHT)))


(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX LIGHT))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'LIGHT)


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'LIGHT)

(SETQ G 'LIGHT)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((LIGHT NIL)))

(SETQ PATH 'LEX)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX LIGHT) (CAT ADJ)))

(SETQ G 'NIL)


** In uavs-ls:

(SETQ I '((CAT NNP) (N ((LEX GRANULE)))))

 G == the whole grammar
(SETQ GA 'ALT)


The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((CAT NNP) (N ((LEX GRANULE)))))

(SETQ G
      '((ALT (((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
```

```
                (N ((CAT NOUN) (LEX ANY))))
              ((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY)))
                      (PATTERN (ADJ NNP)))
                     ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
               (NNP ((CAT NNP))))))))

(SETQ GA 'ALT)

** In uavs-ls:

(SETQ I '((CAT NNP) (N ((LEX GRANULE)))))

(SETQ G
      '((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
        (N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ NONE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input

(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'NONE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NONE NIL)))

(SETQ PATH 'ADJ)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((ADJ NONE) (CAT NNP) (N ((LEX GRANULE)))))

(SETQ G
      '((PP NONE) (TAIL NONE) (PATTERN (N POUND))
        (N ((CAT NOUN) (LEX ANY)))))
```

```
(SETQ GA 'PP)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(PP NONE))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same  PATH was found in the input

(SETQ IVAL 'NIL)


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'NONE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NONE NIL)))

(SETQ PATH 'PP)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((PP NONE) (ADJ NONE) (CAT NNP) (N ((LEX GRANULE)))))

(SETQ G '((TAIL NONE) (PATTERN (N POUND)) (N ((CAT NOUN) (LEX ANY)))))


(SETQ GA 'TAIL)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(TAIL NONE))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input

(SETQ IVAL 'NIL)


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
```

```
** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'NONE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NONE NIL)))

(SETQ PATH 'TAIL)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((TAIL NONE) (PP NONE) (ADJ NONE) (CAT NNP)
                      (N ((LEX GRANULE)))))

(SETQ G '((PATTERN (N POUND)) (N ((CAT NOUN) (LEX ANY)))))


(SETQ GA 'PATTERN)


** In uavs-ls:

(SETQ I
      '((PATTERN (N POUND)) (TAIL NONE) (PP NONE) (ADJ NONE) (CAT NNP)
        (N ((LEX GRANULE)))))

(SETQ G '((N ((CAT NOUN) (LEX ANY)))))


(SETQ GA 'N)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(N ((CAT NOUN) (LEX ANY))))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX GRANULE)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX GRANULE)))

(SETQ G '((CAT NOUN) (LEX ANY)))


(SETQ GA 'CAT)
```

```
 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT NOUN))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same  PATH was found in the input

(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ G 'NOUN)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NOUN NIL)))

(SETQ PATH 'CAT)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT NOUN) (LEX GRANULE)))

(SETQ G '((LEX ANY)))


(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX ANY))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'GRANULE)


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:
```

```
(SETQ I 'GRANULE)

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((GRANULE NIL)))

(SETQ PATH 'LEX)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX GRANULE) (CAT NOUN)))

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX GRANULE) (CAT NOUN)) NIL)))

(SETQ PATH 'N)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((N ((LEX GRANULE) (CAT NOUN))) (PATTERN (N POUND)) (TAIL NONE)
        (PP NONE) (ADJ NONE) (CAT NNP)))

(SETQ G 'NIL)


Alternative succeeded
(SETQ RESULT-0
      '((((N ((LEX GRANULE) (CAT NOUN))) (PATTERN (N POUND))
          (TAIL NONE)
          (PP NONE) (ADJ NONE) (CAT NNP))
         NIL)))


about to quit alt-h-ls
** In uavs-ls:

(SETQ I '((LEX GRANULE) (CAT NOUN)))

 G == the whole grammar
(SETQ GA 'ALT)


The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((LEX GRANULE) (CAT NOUN)))

(SETQ G
      '((ALT (((LEX TELEPHONE)) ((LEX RECEIVER)) ((LEX KIND))
              ((LEX DIAPHRAGM)) ((LEX GRANULE)) ((LEX INTENSITY))
```

```
((LEX MOLECULE)) ((LEX PERSON)) ((LEX POLE))
((LEX RESISTANCE)) ((LEX TRANSMITTER)) ((LEX WAVE))))))

(SETQ GA 'ALT)

lexical level

(SETQ I '((LEX GRANULE) (CAT NOUN)))

(SETQ I-LEX '(LEX GRANULE))

(SETQ G-LEX '((LEX GRANULE)))

** In uavs-ls:

(SETQ I '((LEX GRANULE) (CAT NOUN)))

(SETQ G '((LEX GRANULE)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX GRANULE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'GRANULE)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'GRANULE)

(SETQ G 'GRANULE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((GRANULE ((TAIL NONE) (PP NONE) (ADJ NONE)))))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX GRANULE) (CAT NOUN)))

(SETQ G 'NIL)
```

(THE SMALL NEW LIGHT GRANULE!.)

VII. Trace of the unification of a simplified input with the grammar shown in Appendix I

. Trace of the unification of a simplified input with grammar shown in Appendix I

```
% Tracing variable inside-tracing set to True.
% Tracing the unification process on a simple input, with the grammar
% shown in the appendix.

% Now, the input is not completely specified, so that LOOSE is used.

(setq inside-tracing t)

(setq x1
      '((cat s) (verb ((v === compress)))
        (prot ((article === def) (n === diaphragm)))
        (goal ((number plur) (article === def)
               (adj === small) (n === granule)))))

(pp x1)
(unify x1 grammar)

(SETQ X1
      '((CAT S) (VERB ((V === COMPRESS)))
        (PROT ((ARTICLE === DEF) (N === DIAPHRAGM)))
        (GOAL ((NUMBER PLUR) (ARTICLE === DEF) (ADJ === SMALL)
               (N === GRANULE)))))

T
T

** In uavs-ls:

(SETQ I
      '((CAT S) (VERB ((V ((LEX COMPRESS)))))
        (PROT ((ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))
        (GOAL ((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
               (N ((LEX GRANULE)))))))

(SETQ LOOSE 'NIL)


 G == the whole grammar
(SETQ GA 'ALT)


The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I
      '((CAT S) (VERB ((V ((LEX COMPRESS)))))
        (PROT ((ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))
        (GOAL ((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
               (N ((LEX GRANULE)))))))

(SETQ LOOSE 'NIL)


(SETQ G
 '((ALT (((VERB ((VOICE ACTIVE))) (PROT ANY) (PROT ((CAT NP)))
          (PATTERN (PROT DOTS VERB DOTS POUND))
          (VERB ((CAT VERB-GROUP) (NUMBER (↑ PROT NUMBER))))
          (ALT (((GOAL NONE))
                ((GOAL ANY) (GOAL ((CAT NP)))
                            (PATTERN (DOTS GOAL POUND))))))
```

```
          ((ALT (((VERB ((VOICE PASSIVE))) (PROT NONE))
                  ((VERB ((VOICE ANY) (VOICE PASSIVE))) (PROT ANY)
                   (PROT ((CAT NP)))
                   (BY-OBJ ((CAT PP) (PREP ((LEX BY))) (NP (↑ PROT))))
                   (PATTERN (DOTS VERB BY-OBJ DOTS POUND)))))
           (VERB ((CAT VERB-GROUP)))
           (ALT (((GOAL NONE) (PATTERN (DOTS VERB DOTS))
                  (VERB (NUMBER SING)))
                 ((GOAL ANY) (GOAL ((CAT NP)))
                  (PATTERN (GOAL DOTS VERB DOTS POUND))
                  (VERB ((NUMBER (↑ GOAL NUMBER))))))))))

(SETQ GA 'ALT)

** In uavs-ls:

(SETQ I
      '((CAT S) (VERB ((V ((LEX COMPRESS)))))
        (PROT ((ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))
        (GOAL ((NUMBER PLUR)
               (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
               (N ((LEX GRANULE))))))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((VERB ((VOICE ACTIVE))) (PROT ANY) (PROT ((CAT NP)))
        (PATTERN (PROT DOTS VERB DOTS POUND))
        (VERB ((CAT VERB-GROUP) (NUMBER (↑ PROT NUMBER))))
        (ALT (((GOAL NONE))
              ((GOAL ANY) (GOAL ((CAT NP)))
                          (PATTERN (DOTS GOAL POUND)))))))

(SETQ GA 'VERB)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(VERB ((VOICE ACTIVE))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((V ((LEX COMPRESS)))))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((V ((LEX COMPRESS)))))

(SETQ LOOSE 'NIL)
```

```
(SETQ G '((VOICE ACTIVE)))

(SETQ GA 'VOICE)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(VOICE ACTIVE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE 'NIL)

(SETQ G 'ACTIVE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((ACTIVE NIL)))

(SETQ PATH 'VOICE)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((VOICE ACTIVE) (V ((LEX COMPRESS)))))

(SETQ LOOSE 'NIL)

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((VOICE ACTIVE) (V ((LEX COMPRESS)))) NIL)))

(SETQ PATH 'VERB)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
```

```
      '((VERB ((VOICE ACTIVE) (V ((LEX COMPRESS)))))  (CAT S)
        (PROT ((ARTICLE ((LEX DEF)))  (N ((LEX DIAPHRAGM)))))
        (GOAL ((NUMBER PLUR)  (ARTICLE ((LEX DEF)))  (ADJ ((LEX SMALL)))
               (N ((LEX GRANULE)))))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((PROT ANY)  (PROT ((CAT NP)))
                    (PATTERN (PROT DOTS VERB DOTS POUND))
        (VERB ((CAT VERB-GROUP)  (NUMBER (↑ PROT NUMBER))))
        (ALT (((GOAL NONE))
              ((GOAL ANY)  (GOAL ((CAT NP)))
                           (PATTERN (DOTS GOAL POUND)))))))

(SETQ GA 'PROT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(PROT ANY))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))

(SETQ LOOSE 'NIL)

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))) NIL)))

(SETQ PATH 'PROT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((PROT ((ARTICLE ((LEX DEF)))  (N ((LEX DIAPHRAGM)))))
        (VERB ((VOICE ACTIVE)  (V ((LEX COMPRESS)))))  (CAT S)
        (GOAL ((NUMBER PLUR)  (ARTICLE ((LEX DEF)))
                              (ADJ ((LEX SMALL)))
               (N ((LEX GRANULE)))))))
```

```
(SETQ LOOSE 'NIL)

(SETQ G
      '((PROT ((CAT NP))) (PATTERN (PROT DOTS VERB DOTS POUND))
        (VERB ((CAT VERB-GROUP) (NUMBER (↑ PROT NUMBER))))
        (ALT (((GOAL NONE))
              ((GOAL ANY) (GOAL ((CAT NP)))
                          (PATTERN (DOTS GOAL POUND)))))))

(SETQ GA 'PROT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(PROT ((CAT NP))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))

(SETQ LOOSE 'NIL)

(SETQ G '((CAT NP)))

(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT NP))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:
```

```
(SETQ I 'NIL)

(SETQ LOOSE 'NIL)

(SETQ G 'NP)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NP NIL)))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))

(SETQ LOOSE 'NIL)

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS
      '((((CAT NP) (ARTICLE ((LEX DEF)))
                   (N ((LEX DIAPHRAGM)))) NIL)))

(SETQ PATH 'PROT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((PROT ((CAT NP) (ARTICLE ((LEX DEF)))
                        (N ((LEX DIAPHRAGM)))))
        (VERB ((VOICE ACTIVE) (V ((LEX COMPRESS))))) (CAT S)
        (GOAL ((NUMBER PLUR) (ARTICLE ((LEX DEF)))
                             (ADJ ((LEX SMALL)))
                             (N ((LEX GRANULE))))))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((PATTERN (PROT DOTS VERB DOTS POUND))
        (VERB ((CAT VERB-GROUP) (NUMBER (↑ PROT NUMBER))))
        (ALT (((GOAL NONE))
              ((GOAL ANY) (GOAL ((CAT NP)))
                          (PATTERN (DOTS GOAL POUND)))))))

(SETQ GA 'PATTERN)

** In uavs-ls:
```

```
(SETQ I
      '((PATTERN (PROT DOTS VERB DOTS POUND))
        (PROT ((CAT NP) (ARTICLE ((LEX DEF)))
                        (N ((LEX DIAPHRAGM)))))
        (VERB ((VOICE ACTIVE) (V ((LEX COMPRESS))))) (CAT S)
        (GOAL ((NUMBER PLUR)
               (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
               (N ((LEX GRANULE)))))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((VERB ((CAT VERB-GROUP) (NUMBER (↑ PROT NUMBER))))
        (ALT (((GOAL NONE))
              ((GOAL ANY) (GOAL ((CAT NP)))
                          (PATTERN (DOTS GOAL POUND)))))))

(SETQ GA 'VERB)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(VERB ((CAT VERB-GROUP) (NUMBER (↑ PROT NUMBER)))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((VOICE ACTIVE) (V ((LEX COMPRESS)))))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((VOICE ACTIVE) (V ((LEX COMPRESS)))))

(SETQ LOOSE 'NIL)

(SETQ G '((CAT VERB-GROUP) (NUMBER (↑ PROT NUMBER))))

(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT VERB-GROUP))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input
```

```
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE 'NIL)

(SETQ G 'VERB-GROUP)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((VERB-GROUP NIL)))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT VERB-GROUP) (VOICE ACTIVE) (V ((LEX COMPRESS)))))

(SETQ LOOSE 'NIL)

(SETQ G '((NUMBER (↑ PROT NUMBER))))

(SETQ GA 'NUMBER)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(NUMBER (↑ PROT NUMBER)))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NONE NIL)))

(SETQ PATH 'NUMBER)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
                      (V ((LEX COMPRESS)))))
```

```
(SETQ LOOSE 'NIL)

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS
      '((((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
          (V ((LEX COMPRESS))))
        NIL)))

(SETQ PATH 'VERB)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((VERB ((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
               (V ((LEX COMPRESS)))))
        (PATTERN (PROT DOTS VERB DOTS POUND))
        (PROT ((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))
        (CAT S)
        (GOAL ((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
               (N ((LEX GRANULE)))))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((ALT (((GOAL NONE))
              ((GOAL ANY) (GOAL ((CAT NP)))
                          (PATTERN (DOTS GOAL POUND)))))))

(SETQ GA 'ALT)

** In uavs-ls:

(SETQ I
      '((VERB ((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
               (V ((LEX COMPRESS)))))
        (PATTERN (PROT DOTS VERB DOTS POUND))
        (PROT ((CAT NP) (ARTICLE ((LEX DEF)))
                        (N ((LEX DIAPHRAGM)))))
        (CAT S)
        (GOAL ((NUMBER PLUR) (ARTICLE ((LEX DEF)))
                             (ADJ ((LEX SMALL)))
               (N ((LEX GRANULE)))))))

(SETQ LOOSE 'NIL)

(SETQ G '((GOAL NONE)))

(SETQ GA 'GOAL)

 Calling av-h-ls
 *** In av-h-ls
```

```
******* Attribute-value pair from the grammar:

(SETQ GAV '(GOAL NONE))


*** Calling cntrl-av-h-ls to unify the
input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was found in the input
Value part of the pair found in the input:
(SETQ IVAL
      '((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
        (N ((LEX GRANULE)))))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I
      '((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
        (N ((LEX GRANULE)))))

(SETQ LOOSE 'NIL)


(SETQ G 'NONE)

 Out of uavs-ls
Unifying the 2 values failed

 returning the result of a call to uavs-many-ls

 Alternative has failed...
** In uavs-ls:

(SETQ I
      '((VERB ((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
               (V ((LEX COMPRESS)))))
        (PATTERN (PROT DOTS VERB DOTS POUND))
        (PROT ((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))
        (CAT S)
        (GOAL ((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
               (N ((LEX GRANULE)))))))

(SETQ LOOSE 'NIL)


(SETQ G '((GOAL ANY) (GOAL ((CAT NP))) (PATTERN (DOTS GOAL POUND))))


(SETQ GA 'GOAL)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(GOAL ANY))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar
```

```
In part 1 of the condition statement in CNTRL-AV-H-LS:
 a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL
      '((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
        (N ((LEX GRANULE)))))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I
      '((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
        (N ((LEX GRANULE)))))

(SETQ LOOSE 'NIL)

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS
      '((((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
          (N ((LEX GRANULE))))
        NIL)))

(SETQ PATH 'GOAL)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((GOAL ((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
               (N ((LEX GRANULE)))))
        (VERB ((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
               (V ((LEX COMPRESS)))))
        (PATTERN (PROT DOTS VERB DOTS POUND))
        (PROT ((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))
        (CAT S)))

(SETQ LOOSE 'NIL)

(SETQ G '((GOAL ((CAT NP))) (PATTERN (DOTS GOAL POUND))))

(SETQ GA 'GOAL)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(GOAL ((CAT NP))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
```

```
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL
      '((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
        (N ((LEX GRANULE)))))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I
      '((NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
        (N ((LEX GRANULE)))))

(SETQ LOOSE 'NIL)

(SETQ G '((CAT NP)))

(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT NP))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE 'NIL)

(SETQ G 'NP)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NP NIL)))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((CAT NP) (NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
```

```
        (N ((LEX GRANULE)))))
 (SETQ LOOSE 'NIL)

 (SETQ G 'NIL)
 Out of uavs-ls
 Success: will enrich the input with
 (SETQ RESULTS
       '((((CAT NP) (NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
           (N ((LEX GRANULE))))
         NIL)))
 (SETQ PATH 'GOAL)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
(SETQ I
      '((GOAL ((CAT NP) (NUMBER PLUR) (ARTICLE ((LEX DEF)))
               (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))
        (VERB ((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
               (V ((LEX COMPRESS)))))
        (PATTERN (PROT DOTS VERB DOTS POUND))
        (PROT ((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))
        (CAT S)))
(SETQ LOOSE 'NIL)

(SETQ G '((PATTERN (DOTS GOAL POUND))))

(SETQ GA 'PATTERN)

** In uavs-ls:
(SETQ I
      '((PATTERN (PROT VERB GOAL POUND))
        (GOAL ((CAT NP) (NUMBER PLUR) (ARTICLE ((LEX DEF)))
               (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))
        (VERB ((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
               (V ((LEX COMPRESS)))))
        (PROT ((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))
        (CAT S)))
(SETQ LOOSE 'NIL)

(SETQ G 'NIL)

Alternative succeeded
(SETQ RESULT-0
      '((((PATTERN (PROT VERB GOAL POUND))
          (GOAL ((CAT NP) (NUMBER PLUR) (ARTICLE ((LEX DEF)))
                 (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))
          (VERB ((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
                 (V ((LEX COMPRESS)))))
          (PROT ((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))
```

```
          (CAT S))
         NIL)))

about to quit alt-h-ls
Alternative succeeded
(SETQ RESULT-0
      '((((PATTERN (PROT VERB GOAL POUND))
          (GOAL ((CAT NP) (NUMBER PLUR) (ARTICLE ((LEX DEF)))
                 (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))
          (VERB ((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
                 (V ((LEX COMPRESS)))))
          (PROT ((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))
          (CAT S))
         NIL)))

about to quit alt-h-ls
** In uavs-ls:

(SETQ I '((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))

(SETQ LOOSE 'NIL)

 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((ALT (((ARTICLE NONE) (PATTERN (DOTS NNP DOTS)))
              ((ARTICLE ANY) (ARTICLE ((CAT ARTICLE) (LEX ANY)))
               (PATTERN (DOTS ARTICLE NNP DOTS)))))
        (NNP ((CAT NNP)))))

(SETQ GA 'ALT)

** In uavs-ls:

(SETQ I '((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))

(SETQ LOOSE 'NIL)

(SETQ G '((ARTICLE NONE) (PATTERN (DOTS NNP DOTS)) (NNP ((CAT NNP)))))

(SETQ GA 'ARTICLE)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:
```

```
(SETQ GAV '(ARTICLE NONE))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX DEF)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX DEF)))

(SETQ LOOSE 'NIL)


(SETQ G 'NONE)

 Out of uavs-ls
Unifying the 2 values failed

 returning the result of a call to uavs-many-ls

 Alternative has failed...
** In uavs-ls:

(SETQ I '((CAT NP) (ARTICLE ((LEX DEF))) (N ((LEX DIAPHRAGM)))))

(SETQ LOOSE 'NIL)


(SETQ G
      '((ARTICLE ANY) (ARTICLE ((CAT ARTICLE) (LEX ANY)))
        (PATTERN (DOTS ARTICLE NNP DOTS)) (NNP ((CAT NNP)))))


(SETQ GA 'ARTICLE)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ARTICLE ANY))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX DEF)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX DEF)))
```

```
(SETQ LOOSE 'NIL)

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX DEF)) NIL)))

(SETQ PATH 'ARTICLE)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((ARTICLE ((LEX DEF))) (CAT NP) (N ((LEX DIAPHRAGM)))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((ARTICLE ((CAT ARTICLE) (LEX ANY)))
        (PATTERN (DOTS ARTICLE NNP DOTS)) (NNP ((CAT NNP)))))

(SETQ GA 'ARTICLE)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ARTICLE ((CAT ARTICLE) (LEX ANY))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX DEF)))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX DEF)))

(SETQ LOOSE 'NIL)

(SETQ G '((CAT ARTICLE) (LEX ANY)))

(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:
```

```
(SETQ GAV '(CAT ARTICLE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same  PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE 'NIL)

(SETQ G 'ARTICLE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((ARTICLE NIL)))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT ARTICLE) (LEX DEF)))

(SETQ LOOSE 'NIL)

(SETQ G '((LEX ANY)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX ANY))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'DEF)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
```

```
** In uavs-ls:
(SETQ I 'DEF)
(SETQ LOOSE 'NIL)

(SETQ G 'ANY)
 Out of uavs-ls
 Success: will enrich the input with
(SETQ RESULTS '((DEF NIL)))
(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
(SETQ I '((LEX DEF) (CAT ARTICLE)))
(SETQ LOOSE 'NIL)

(SETQ G 'NIL)
 Out of uavs-ls
 Success: will enrich the input with
(SETQ RESULTS '((((LEX DEF) (CAT ARTICLE)) NIL)))
(SETQ PATH 'ARTICLE)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
(SETQ I
      '((ARTICLE ((LEX DEF) (CAT ARTICLE))) (CAT NP)
                                            (N ((LEX DIAPHRAGM)))))
(SETQ LOOSE 'NIL)

(SETQ G '((PATTERN (DOTS ARTICLE NNP DOTS)) (NNP ((CAT NNP)))))

(SETQ GA 'PATTERN)

** In uavs-ls:
(SETQ I
      '((PATTERN (DOTS ARTICLE NNP DOTS))
        (ARTICLE ((LEX DEF) (CAT ARTICLE))) (CAT NP)
        (N ((LEX DIAPHRAGM)))))
(SETQ LOOSE 'NIL)

(SETQ G '((NNP ((CAT NNP)))))
```

```
(SETQ GA 'NNP)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:
(SETQ GAV '(NNP ((CAT NNP))))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE 'NIL)

(SETQ G '((CAT NNP)))

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((CAT NNP)) NIL)))

(SETQ PATH 'NNP)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((NNP ((CAT NNP))) (PATTERN (DOTS ARTICLE NNP DOTS))
        (ARTICLE ((LEX DEF) (CAT ARTICLE))) (CAT NP)
        (N ((LEX DIAPHRAGM)))))

(SETQ LOOSE 'NIL)


(SETQ G 'NIL)


Alternative succeeded
(SETQ RESULT-0
      '((((NNP ((CAT NNP))) (PATTERN (DOTS ARTICLE NNP DOTS))
          (ARTICLE ((LEX DEF) (CAT ARTICLE))) (CAT NP)
          (N ((LEX DIAPHRAGM))))
         NIL)))


about to quit alt-h-ls
** In uavs-ls:
```

```
(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G '((ALT (((LEX INDEF)) ((LEX DEF))))))

(SETQ GA 'ALT)

lexical level

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ I-LEX '(LEX DEF))

(SETQ G-LEX '((LEX DEF)))

** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G '((LEX DEF)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX DEF))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'DEF)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:
```

```
(SETQ I 'DEF)
(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G 'DEF)
 Out of uavs-ls
 Success: will enrich the input with
(SETQ RESULTS '((DEF ((N ((LEX DIAPHRAGM))))))
(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
(SETQ I '((LEX DEF) (CAT ARTICLE)))
(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G 'NIL)

** In uavs-ls:
(SETQ I '((CAT NNP)))
(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:
(SETQ I '((CAT NNP)))
(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G
      '((ALT (((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
               (N ((CAT NOUN) (LEX ANY))))
              ((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY)))
                      (PATTERN (ADJ NNP)))
                     ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
               (NNP ((CAT NNP)))))))

(SETQ GA 'ALT)

** In uavs-ls:
(SETQ I '((CAT NNP)))
(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))
```

```
(SETQ G
      '((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
        (N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ NONE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G 'NONE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NONE ((N ((LEX DIAPHRAGM)))))))

(SETQ PATH 'ADJ)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((ADJ NONE) (CAT NNP)))

(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G
      '((PP NONE) (TAIL NONE) (PATTERN (N POUND))
        (N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'PP)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:
```

```
(SETQ GAV '(PP NONE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
 no pair with same   PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G 'NONE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NONE ((N ((LEX DIAPHRAGM)))))))

(SETQ PATH 'PP)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((PP NONE) (ADJ NONE) (CAT NNP)))

(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G '((TAIL NONE) (PATTERN (N POUND)) (N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'TAIL)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(TAIL NONE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
```

```
** In uavs-ls:
(SETQ I 'NIL)
(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G 'NONE)
 Out of uavs-ls
 Success: will enrich the input with
(SETQ RESULTS '((NONE ((N ((LEX DIAPHRAGM)))))))
(SETQ PATH 'TAIL)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
(SETQ I '((TAIL NONE) (PP NONE) (ADJ NONE) (CAT NNP)))
(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G '((PATTERN (N POUND)) (N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'PATTERN)

** In uavs-ls:
(SETQ I '((PATTERN (N POUND)) (TAIL NONE) (PP NONE) (ADJ NONE)
                              (CAT NNP)))
(SETQ LOOSE '((N ((LEX DIAPHRAGM)))))

(SETQ G '((N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'N)
 Calling av-h-ls
 *** In av-h-ls
 ******* Attribute-value pair from the grammar:
(SETQ GAV '(N ((CAT NOUN) (LEX ANY))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar
In part 2 of the condition statement in CNTRL-AV-H-LS:
 no pair with same   PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL '((LEX DIAPHRAGM)))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
** In uavs-ls:
```

```
(SETQ I '((LEX DIAPHRAGM)))

(SETQ LOOSE 'NIL)

(SETQ G '((CAT NOUN) (LEX ANY)))

(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT NOUN))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same  PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE 'NIL)

(SETQ G 'NOUN)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NOUN NIL)))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT NOUN) (LEX DIAPHRAGM)))

(SETQ LOOSE 'NIL)

(SETQ G '((LEX ANY)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:
```

```
(SETQ GAV '(LEX ANY))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
 a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'DIAPHRAGM)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'DIAPHRAGM)

(SETQ LOOSE 'NIL)

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((DIAPHRAGM NIL)))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX DIAPHRAGM) (CAT NOUN)))

(SETQ LOOSE 'NIL)

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX DIAPHRAGM) (CAT NOUN)) NIL)))

(SETQ PATH 'N)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((N ((LEX DIAPHRAGM) (CAT NOUN))) (PATTERN (N POUND))
                                         (TAIL NONE)
        (PP NONE) (ADJ NONE) (CAT NNP)))

(SETQ LOOSE 'NIL)

(SETQ G 'NIL)
```

```
Alternative succeeded
(SETQ RESULT-0
      '((((N ((LEX DIAPHRAGM) (CAT NOUN))) (PATTERN (N POUND))
                                           (TAIL NONE)
         (PP NONE) (ADJ NONE) (CAT NNP))
        NIL)))

about to quit alt-h-ls
** In uavs-ls:

(SETQ I '((LEX DIAPHRAGM) (CAT NOUN)))

(SETQ LOOSE '((TAIL NONE) (PP NONE) (ADJ NONE)))

 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((LEX DIAPHRAGM) (CAT NOUN)))

(SETQ LOOSE '((TAIL NONE) (PP NONE) (ADJ NONE)))

(SETQ G
      '((ALT (((LEX TELEPHONE)) ((LEX RECEIVER)) ((LEX KIND))
              ((LEX DIAPHRAGM)) ((LEX GRANULE)) ((LEX INTENSITY))
              ((LEX MOLECULE)) ((LEX PERSON)) ((LEX POLE))
              ((LEX RESISTANCE)) ((LEX TRANSMITTER)) ((LEX WAVE))))))

(SETQ GA 'ALT)

lexical level

(SETQ I '((LEX DIAPHRAGM) (CAT NOUN)))

(SETQ I-LEX '(LEX DIAPHRAGM))

(SETQ G-LEX '((LEX DIAPHRAGM)))

** In uavs-ls:

(SETQ I '((LEX DIAPHRAGM) (CAT NOUN)))

(SETQ LOOSE '((TAIL NONE) (PP NONE) (ADJ NONE)))

(SETQ G '((LEX DIAPHRAGM)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:
```

```
(SETQ GAV '(LEX DIAPHRAGM))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'DIAPHRAGM)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'DIAPHRAGM)

(SETQ LOOSE '((TAIL NONE) (PP NONE) (ADJ NONE)))

(SETQ G 'DIAPHRAGM)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((DIAPHRAGM ((TAIL NONE) (PP NONE) (ADJ NONE)))))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX DIAPHRAGM) (CAT NOUN)))

(SETQ LOOSE '((TAIL NONE) (PP NONE) (ADJ NONE)))

(SETQ G 'NIL)

** In uavs-ls:

(SETQ I
      '((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
                      (V ((LEX COMPRESS)))))

(SETQ LOOSE 'NIL)

 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I
      '((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
                      (V ((LEX COMPRESS)))))
```

```
(SETQ LOOSE 'NIL)

(SETQ G
      '((ALT (((VOICE ACTIVE) (PATTERN (V DOTS))
               (V ((CAT VERB) (LEX ANY))))
              ((VOICE PASSIVE) (PATTERN (V1 V DOTS))
               (V1 ((CAT VERB) (LEX BE)))
               (V ((CAT VERB) (LEX ANY) (TENSE PASTP))))))
        (ALT (((PP NONE)) ((PP ANY) (PP ((CAT PP)))
                                    (PATTERN (DOTS PP)))))))

(SETQ GA 'ALT)

** In uavs-ls:

(SETQ I
      '((NUMBER NONE) (CAT VERB-GROUP) (VOICE ACTIVE)
                      (V ((LEX COMPRESS)))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((VOICE ACTIVE) (PATTERN (V DOTS)) (V ((CAT VERB) (LEX ANY)))
        (ALT (((PP NONE)) ((PP ANY) (PP ((CAT PP)))
                                    (PATTERN (DOTS PP)))))))

(SETQ GA 'VOICE)

Calling av-h-ls
*** In av-h-ls

******* Attribute-value pair from the grammar:

(SETQ GAV '(VOICE ACTIVE))

*** Calling cntrl-av-h-ls to unify the
input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
 a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'ACTIVE)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'ACTIVE)

(SETQ LOOSE 'NIL)

(SETQ G 'ACTIVE)

Out of uavs-ls

Success: will enrich the input with
(SETQ RESULTS '((ACTIVE NIL)))
```

```
(SETQ PATH 'VOICE)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((VOICE ACTIVE) (NUMBER NONE) (CAT VERB-GROUP)
                       (V ((LEX COMPRESS)))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((PATTERN (V DOTS)) (V ((CAT VERB) (LEX ANY)))
        (ALT (((PP NONE)) ((PP ANY) (PP ((CAT PP)))
                                    (PATTERN (DOTS PP)))))))

(SETQ GA 'PATTERN)

** In uavs-ls:

(SETQ I
      '((PATTERN (V DOTS)) (VOICE ACTIVE) (NUMBER NONE) (CAT VERB-GROUP)
        (V ((LEX COMPRESS)))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((V ((CAT VERB) (LEX ANY)))
        (ALT (((PP NONE)) ((PP ANY) (PP ((CAT PP)))
                                    (PATTERN (DOTS PP)))))))

(SETQ GA 'V)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(V ((CAT VERB) (LEX ANY))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
 a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX COMPRESS)))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX COMPRESS)))

(SETQ LOOSE 'NIL)
```

```
(SETQ G '((CAT VERB) (LEX ANY)))

(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT VERB))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE 'NIL)

(SETQ G 'VERB)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((VERB NIL)))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT VERB) (LEX COMPRESS)))

(SETQ LOOSE 'NIL)

(SETQ G '((LEX ANY)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX ANY))
```

```
*** Calling cntrl-av-h-ls to unify the
input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input    .
Value part of the pair found in the input:
(SETQ IVAL 'COMPRESS)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'COMPRESS)

(SETQ LOOSE 'NIL)

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((COMPRESS NIL)))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX COMPRESS) (CAT VERB)))

(SETQ LOOSE 'NIL)

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX COMPRESS) (CAT VERB)) NIL)))

(SETQ PATH 'V)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((V ((LEX COMPRESS) (CAT VERB))) (PATTERN (V DOTS))
                                        (VOICE ACTIVE)
        (NUMBER NONE) (CAT VERB-GROUP)))

(SETQ LOOSE 'NIL)

(SETQ G
      '((ALT (((PP NONE)) ((PP ANY) (PP ((CAT PP)))
                                    (PATTERN (DOTS PP)))))))

(SETQ GA 'ALT)
```

```
** In uavs-ls:
(SETQ I
      '((V ((LEX COMPRESS) (CAT VERB))) (PATTERN (V DOTS))
        (NUMBER NONE) (CAT VERB-GROUP))) (VOICE ACTIVE)
(SETQ LOOSE 'NIL)

(SETQ G '((PP NONE)))

(SETQ GA 'PP)
 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:
(SETQ GAV '(PP NONE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar
In part 2 of the condition statement in CNTRL-AV-H-LS:
no pair with same   PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
** In uavs-ls:
(SETQ I 'NIL)
(SETQ LOOSE 'NIL)

(SETQ G 'NONE)
 Out of uavs-ls
 Success: will enrich the input with
(SETQ RESULTS '((NONE NIL)))
(SETQ PATH 'PP)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
(SETQ I
      '((PP NONE) (V ((LEX COMPRESS) (CAT VERB))) (PATTERN (V DOTS))
        (VOICE ACTIVE) (NUMBER NONE) (CAT VERB-GROUP)))
(SETQ LOOSE 'NIL)

(SETQ G 'NIL)
```

```
Alternative succeeded
(SETQ RESULT-0
      '((((PP NONE) (V ((LEX COMPRESS) (CAT VERB))) (PATTERN (V DOTS))
          (VOICE ACTIVE) (NUMBER NONE) (CAT VERB-GROUP))
        NIL)))


about to quit alt-h-ls
Alternative succeeded
(SETQ RESULT-0
      '((((PP NONE) (V ((LEX COMPRESS) (CAT VERB))) (PATTERN (V DOTS))
          (VOICE ACTIVE) (NUMBER NONE) (CAT VERB-GROUP))
        NIL)))


about to quit alt-h-ls
** In uavs-ls:

(SETQ I '((LEX COMPRESS) (CAT VERB)))

(SETQ LOOSE '((PP NONE) (NUMBER NONE)))


 G == the whole grammar
(SETQ GA 'ALT)


The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((LEX COMPRESS) (CAT VERB)))

(SETQ LOOSE '((PP NONE) (NUMBER NONE)))


(SETQ G
      '((ALT (((LEX ATTRACT)) ((LEX CAUSE)) ((LEX COMPRESS))
                              ((LEX HIT))
              ((LEX INCREASE)) ((LEX MOVE)) ((LEX SPEAK))
              ((LEX VIBRATE))
             ((LEX VARY)
              (PRESENT (SING (FIRST VARY) (SECOND VARY)
                             (THIRD VARIES))
                       (PLUR (FIRST VARY) (SECOND VARY)
                             (THIRD VARY)))
              (PASTP (SING (FIRST VARIED) (SECOND VARIED)
                           (THIRD VARIED))
                     (PLUR (FIRST VARIED) (SECOND VARIED)
                           (THIRD VARIED))))
             ((LEX HAVE)
              (PRESENT (SING (FIRST HAVE) (SECOND HAVE)
                             (THIRD HAS))
                       (PLUR (FIRST HAVE) (SECOND HAVE)
                             (THIRD HAVE))))
             ((LEX BE)
              (PRESENT (SING (FIRST AM) (SECOND ARE) (THIRD IS))
                       (PLUR (FIRST ARE) (SECOND ARE)
                             (THIRD ARE)))))))))

(SETQ GA 'ALT)
```

```
lexical level
(SETQ I '((LEX COMPRESS) (CAT VERB)))
(SETQ I-LEX '(LEX COMPRESS))
(SETQ G-LEX '((LEX COMPRESS)))

** In uavs-ls:
(SETQ I '((LEX COMPRESS) (CAT VERB)))
(SETQ LOOSE '((PP NONE) (NUMBER NONE)))

(SETQ G '((LEX COMPRESS)))

(SETQ GA 'LEX)
 Calling av-h-ls
 *** In av-h-ls
 ******* Attribute-value pair from the grammar:
(SETQ GAV '(LEX COMPRESS))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar
In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'COMPRESS)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
** In uavs-ls:
(SETQ I 'COMPRESS)
(SETQ LOOSE '((PP NONE) (NUMBER NONE)))

(SETQ G 'COMPRESS)
 Out of uavs-ls
 Success: will enrich the input with
(SETQ RESULTS '((COMPRESS ((PP NONE) (NUMBER NONE)))))
(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
(SETQ I '((LEX COMPRESS) (CAT VERB)))
(SETQ LOOSE '((PP NONE) (NUMBER NONE)))
```

```
(SETQ G 'NIL)

** In uavs-ls:

(SETQ I
      '((CAT NP) (NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
        (N ((LEX GRANULE)))))

(SETQ LOOSE 'NIL)

 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based     CAT in the input;
 UAVS-LS is now called recursively with  thi.  :ategory
** In uavs-ls:

(SETQ I
      '((CAT NP) (NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
        (N ((LEX GRANULE)))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((ALT (((ARTICLE NONE) (PATTERN (DOTS NNP DOTS)))
              ((ARTICLE ANY) (ARTICLE ((CAT ARTICLE) (LEX ANY)))
               (PATTERN (DOTS ARTICLE NNP DOTS)))))
        (NNP ((CAT NNP)))))

(SETQ GA 'ALT)

** In uavs-ls:

(SETQ I
      '((CAT NP) (NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
        (N ((LEX GRANULE)))))

(SETQ LOOSE 'NIL)

(SETQ G '((ARTICLE NONE) (PATTERN (DOTS NNP DOTS)) (NNP ((CAT NNP)))))

(SETQ GA 'ARTICLE)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ARTICLE NONE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
```

```
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX DEF)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX DEF)))

(SETQ LOOSE 'NIL)


(SETQ G 'NONE)

 Out of uavs-ls
Unifying the 2 values failed

 returning the result of a call to uavs-many-ls

 Alternative has failed...
** In uavs-ls:

(SETQ I
      '((CAT NP) (NUMBER PLUR) (ARTICLE ((LEX DEF))) (ADJ ((LEX SMALL)))
        (N ((LEX GRANULE)))))

(SETQ LOOSE 'NIL)


(SETQ G
      '((ARTICLE ANY) (ARTICLE ((CAT ARTICLE) (LEX ANY)))
        (PATTERN (DOTS ARTICLE NNP DOTS)) (NNP ((CAT NNP)))))


(SETQ GA 'ARTICLE)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ARTICLE ANY))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX DEF)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX DEF)))

(SETQ LOOSE 'NIL)


(SETQ G 'ANY)
```

```
 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX DEF)) NIL)))

(SETQ PATH 'ARTICLE)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((ARTICLE ((LEX DEF))) (CAT NP) (NUMBER PLUR) (ADJ ((LEX SMALL)))
        (N ((LEX GRANULE)))))

(SETQ LOOSE 'NIL)

(SETQ G
      '((ARTICLE ((CAT ARTICLE) (LEX ANY)))
        (PATTERN (DOTS ARTICLE NNP DOTS)) (NNP ((CAT NNP)))))

(SETQ GA 'ARTICLE)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ARTICLE ((CAT ARTICLE) (LEX ANY))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX DEF)))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX DEF)))

(SETQ LOOSE 'NIL)

(SETQ G '((CAT ARTICLE) (LEX ANY)))

(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT ARTICLE))
```

```
*** Calling cntrl-av-h-ls to unify the
input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS: no pair with same
 PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE 'NIL)

(SETQ G 'ARTICLE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((ARTICLE NIL)))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT ARTICLE) (LEX DEF)))

(SETQ LOOSE 'NIL)

(SETQ G '((LEX ANY)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX ANY))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
 a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'DEF)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'DEF)
```

```
(SETQ LOOSE 'NIL)

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((DEF NIL)))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ LOOSE 'NIL)

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX DEF) (CAT ARTICLE)) NIL)))

(SETQ PATH 'ARTICLE)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((ARTICLE ((LEX DEF) (CAT ARTICLE))) (CAT NP) (NUMBER PLUR)
        (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))

(SETQ LOOSE 'NIL)

(SETQ G '((PATTERN (DOTS ARTICLE NNP DOTS)) (NNP ((CAT NNP)))))

(SETQ GA 'PATTERN)

** In uavs-ls:

(SETQ I
      '((PATTERN (DOTS ARTICLE NNP DOTS))
        (ARTICLE ((LEX DEF) (CAT ARTICLE))) (CAT NP) (NUMBER PLUR)
        (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))

(SETQ LOOSE 'NIL)

(SETQ G '((NNP ((CAT NNP)))))

(SETQ GA 'NNP)
```

```
 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(NNP ((CAT NNP))))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS: no pair with same
 PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE 'NIL)


(SETQ G '((CAT NNP)))

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((CAT NNP)) NIL)))

(SETQ PATH 'NNP)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((NNP ((CAT NNP))) (PATTERN (DOTS ARTICLE NNP DOTS))
        (ARTICLE ((LEX DEF) (CAT ARTICLE))) (CAT NP) (NUMBER PLUR)
        (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))

(SETQ LOOSE 'NIL)


(SETQ G 'NIL)


Alternative succeeded
(SETQ RESULT-0
      '((((NNP ((CAT NNP))) (PATTERN (DOTS ARTICLE NNP DOTS))
          (ARTICLE ((LEX DEF) (CAT ARTICLE))) (CAT NP) (NUMBER PLUR)
          (ADJ ((LEX SMALL))) (N ((LEX GRANULE))))
         NIL)))


about to quit alt-h-ls
** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ LOOSE '((NUMBER PLUR) (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))
```

```
 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ LOOSE '((NUMBER PLUR) (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))

(SETQ G '((ALT (((LEX INDEF)) ((LEX DEF))))))

(SETQ GA 'ALT)

lexical level

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ I-LEX '(LEX DEF))

(SETQ G-LEX '((LEX DEF)))

** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ LOOSE '((NUMBER PLUR) (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))

(SETQ G '((LEX DEF)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX DEF))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
 a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'DEF)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'DEF)
```

```
(SETQ LOOSE '((NUMBER PLUR) (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))

(SETQ G 'DEF)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS
      '((DEF ((NUMBER PLUR) (ADJ ((LEX SMALL))) (N ((LEX GRANULE))))))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX DEF) (CAT ARTICLE)))

(SETQ LOOSE '((NUMBER PLUR) (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))

(SETQ G 'NIL)

** In uavs-ls:

(SETQ I '((CAT NNP)))

(SETQ LOOSE '((NUMBER PLUR) (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))

 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((CAT NNP)))

(SETQ LOOSE '((NUMBER PLUR) (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))

(SETQ G
      '((ALT (((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
               (N ((CAT NOUN) (LEX ANY))))
              ((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY)))
                      (PATTERN (ADJ NNP)))
                     ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
               (NNP ((CAT NNP)))))))))

(SETQ GA 'ALT)

** In uavs-ls:

(SETQ I '((CAT NNP)))

(SETQ LOOSE '((NUMBER PLUR) (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))
```

```
(SETQ G
      '((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
        (N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ NONE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS: no pair with same
 PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL '((LEX SMALL)))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX SMALL)))

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G 'NONE)

 Out of uavs-ls
Unifying the 2 values failed

 returning the result of a call to uavs-many-ls

 Alternative has failed...
** In uavs-ls:

(SETQ I '((CAT NNP)))

(SETQ LOOSE '((NUMBER PLUR) (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))

(SETQ G
      '((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY))) (PATTERN (ADJ NNP)))
              ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
        (NNP ((CAT NNP)))))

(SETQ GA 'ALT)

** In uavs-ls:

(SETQ I '((CAT NNP)))

(SETQ LOOSE '((NUMBER PLUR) (ADJ ((LEX SMALL))) (N ((LEX GRANULE)))))

(SETQ G
```

```
      '((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY))) (PATTERN (ADJ NNP))
        (NNP ((CAT NNP)))))

(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ ANY))


 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS: no pair with same
 PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL '((LEX SMALL)))


** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX SMALL)))

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))


(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX SMALL)) ((NUMBER PLUR) (N ((LEX GRANULE)))))))

(SETQ PATH 'ADJ)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((ADJ ((LEX SMALL))) (CAT NNP)))

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))


(SETQ G
      '((ADJ ((CAT ADJ) (LEX ANY))) (PATTERN (ADJ NNP)) (NNP ((CAT NNP))))


(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ ((CAT ADJ) (LEX ANY))))
```

```
*** Calling cntrl-av-h-ls to unify the
input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL '((LEX SMALL)))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX SMALL)))

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G '((CAT ADJ) (LEX ANY)))

(SETQ GA 'CAT)

Calling av-h-ls
*** In av-h-ls

******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT ADJ))

*** Calling cntrl-av-h-ls to unify the
input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS: no pair with same
 PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G 'ADJ)

Out of uavs-ls

Success: will enrich the input with
(SETQ RESULTS '((ADJ ((NUMBER PLUR) (N ((LEX GRANULE)))))))

(SETQ PATH 'CAT)

returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT ADJ) (LEX SMALL)))
```

```
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G '((LEX ANY)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX ANY))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'SMALL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'SMALL)

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((SMALL ((NUMBER PLUR) (N ((LEX GRANULE)))))))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((LEX SMALL) (CAT ADJ)))

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS
      '((((LEX SMALL) (CAT ADJ)) ((NUMBER PLUR) (N ((LEX GRANULE)))))))

(SETQ PATH 'ADJ)

 returning the result of a call to uavs-many-ls
```

```
** In uavs-ls:
(SETQ I '((ADJ ((LEX SMALL) (CAT ADJ))) (CAT NNP)))
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G '((PATTERN (ADJ NNP)) (NNP ((CAT NNP)))))

(SETQ GA 'PATTERN)

** In uavs-ls:
(SETQ I '((PATTERN (ADJ NNP)) (ADJ ((LEX SMALL) (CAT ADJ))) (CAT NNP)))
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G '((NNP ((CAT NNP)))))

(SETQ GA 'NNP)
 Calling av-h-ls
 *** In av-h-ls
 ******* Attribute-value pair from the grammar:
(SETQ GAV '(NNP ((CAT NNP))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar
In part 2 of the condition statement in CNTRL-AV-H-LS: no pair with same
 PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
** In uavs-ls:
(SETQ I 'NIL)
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G '((CAT NNP)))
 Out of uavs-ls
 Success: will enrich the input with
(SETQ RESULTS '((((CAT NNP)) ((NUMBER PLUR) (N ((LEX GRANULE)))))))
(SETQ PATH 'NNP)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
```

```
(SETQ I
      '((NNP ((CAT NNP))) (PATTERN (ADJ NNP))
        (ADJ ((LEX SMALL) (CAT ADJ))) (CAT NNP)))

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G 'NIL)

Alternative succeeded
(SETQ RESULT-0
      '((((NNP ((CAT NNP))) (PATTERN (ADJ NNP))
          (ADJ ((LEX SMALL) (CAT ADJ))) (CAT NNP))
         ((NUMBER PLUR) (N ((LEX GRANULE)))))))

about to quit alt-h-ls
Alternative succeeded
(SETQ RESULT-0
      '((((NNP ((CAT NNP))) (PATTERN (ADJ NNP))
          (ADJ ((LEX SMALL) (CAT ADJ))) (CAT NNP))
         ((NUMBER PLUR) (N ((LEX GRANULE)))))))

about to quit alt-h-ls
** In uavs-ls:

(SETQ I '((LEX SMALL) (CAT ADJ)))

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((LEX SMALL) (CAT ADJ)))

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G
      '((ALT (((LEX BIG)) ((LEX LIGHT)) ((LEX CURRENT)) ((LEX NEW))
              ((LEX OLD)) ((LEX SMALL)) ((LEX SOUND))))))

(SETQ GA 'ALT)

lexical level

(SETQ I '((LEX SMALL) (CAT ADJ)))

(SETQ I-LEX '(LEX SMALL))

(SETQ G-LEX '((LEX SMALL)))

** In uavs-ls:
```

```
(SETQ I '((LEX SMALL) (CAT ADJ)))
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G '((LEX SMALL)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:
(SETQ GAV '(LEX SMALL))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar
In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'SMALL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
** In uavs-ls:
(SETQ I 'SMALL)
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G 'SMALL)
 Out of uavs-ls
 Success: will enrich the input with
(SETQ RESULTS '((SMALL ((NUMBER PLUR) (N ((LEX GRANULE)))))))
(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
(SETQ I '((LEX SMALL) (CAT ADJ)))
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G 'NIL)

** In uavs-ls:
(SETQ I '((CAT NNP)))
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))
```

```
 G == the whole grammar
(SETQ GA 'ALT)

The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((CAT NNP)))

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G
      '((ALT (((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
               (N ((CAT NOUN) (LEX ANY))))
              ((ALT (((ADJ ANY) (ADJ ((CAT ADJ) (LEX ANY)))
                      (PATTERN (ADJ NNP)))
                     ((PP ANY) (PP ((CAT PP))) (PATTERN (NNP PP)))))
               (NNP ((CAT NNP))))))))

(SETQ GA 'ALT)

** In uavs-ls:

(SETQ I '((CAT NNP)))

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G
      '((ADJ NONE) (PP NONE) (TAIL NONE) (PATTERN (N POUND))
        (N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'ADJ)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(ADJ NONE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS: no pair with same
 PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))
```

```
(SETQ G 'NONE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NONE ((NUMBER PLUR) (N ((LEX GRANULE))))))

(SETQ PATH 'ADJ)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((ADJ NONE) (CAT NNP)))

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G
      '((PP NONE) (TAIL NONE) (PATTERN (N POUND))
        (N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'PP)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(PP NONE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS: no pair with same
 PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G 'NONE)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NONE ((NUMBER PLUR) (N ((LEX GRANULE))))))

(SETQ PATH 'PP)

 returning the result of a call to uavs-many-ls
```

```
** In uavs-ls:
(SETQ I '((PP NONE) (ADJ NONE) (CAT NNP)))
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G '((TAIL NONE) (PATTERN (N POUND)) (N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'TAIL)
 Calling av-h-ls
 *** In av-h-ls
 ******* Attribute-value pair from the grammar:
(SETQ GAV '(TAIL NONE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar
In part 2 of the condition statement in CNTRL-AV-H-LS: no pair with same
 PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
** In uavs-ls:
(SETQ I 'NIL)
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G 'NONE)
 Out of uavs-ls
 Success: will enrich the input with
(SETQ RESULTS '((NONE ((NUMBER PLUR) (N ((LEX GRANULE)))))))
(SETQ PATH 'TAIL)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
(SETQ I '((TAIL NONE) (PP NONE) (ADJ NONE) (CAT NNP)))
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G '((PATTERN (N POUND)) (N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'PATTERN)

** In uavs-ls:
```

```
(SETQ I '((PATTERN (N POUND)) (TAIL NONE) (PP NONE) (ADJ NONE) (CAT NNP)))
(SETQ LOOSE '((NUMBER PLUR) (N ((LEX GRANULE)))))

(SETQ G '((N ((CAT NOUN) (LEX ANY)))))

(SETQ GA 'N)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(N ((CAT NOUN) (LEX ANY))))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS: no pair with same
 PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL '((LEX GRANULE)))

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I '((LEX GRANULE)))

(SETQ LOOSE '((NUMBER PLUR)))

(SETQ G '((CAT NOUN) (LEX ANY)))

(SETQ GA 'CAT)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(CAT NOUN))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 2 of the condition statement in CNTRL-AV-H-LS: no pair with same
 PATH was found in the input
 Looks into LOOSE for it...
(SETQ IVAL 'NIL)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'NIL)

(SETQ LOOSE '((NUMBER PLUR)))
```

```
(SETQ G 'NOUN)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((NOUN ((NUMBER PLUR)))))

(SETQ PATH 'CAT)

 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I '((CAT NOUN) (LEX GRANULE)))

(SETQ LOOSE '((NUMBER PLUR)))

(SETQ G '((LEX ANY)))

(SETQ GA 'LEX)

 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:

(SETQ GAV '(LEX ANY))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar

In part 1 of the condition statement in CNTRL-AV-H-LS:
a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'GRANULE)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values

** In uavs-ls:

(SETQ I 'GRANULE)

(SETQ LOOSE '((NUMBER PLUR)))

(SETQ G 'ANY)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((GRANULE ((NUMBER PLUR)))))

(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls

** In uavs-ls:
```

```
(SETQ I '((LEX GRANULE) (CAT NOUN)))

(SETQ LOOSE '((NUMBER PLUR)))


(SETQ G 'NIL)

 Out of uavs-ls

 Success: will enrich the input with
(SETQ RESULTS '((((LEX GRANULE) (CAT NOUN)) ((NUMBER PLUR)))))

(SETQ PATH 'N)


 returning the result of a call to uavs-many-ls

** In uavs-ls:

(SETQ I
      '((N ((LEX GRANULE) (CAT NOUN))) (PATTERN (N POUND)) (TAIL NONE)
        (PP NONE) (ADJ NONE) (CAT NNP)))

(SETQ LOOSE '((NUMBER PLUR)))


(SETQ G 'NIL)


Alternative succeeded
(SETQ RESULT-0
      '((((N ((LEX GRANULE) (CAT NOUN))) (PATTERN (N POUND)) (TAIL NONE)
          (PP NONE) (ADJ NONE) (CAT NNP))
         ((NUMBER PLUR)))))


about to quit alt-h-ls
** In uavs-ls:

(SETQ I '((LEX GRANULE) (CAT NOUN)))

(SETQ LOOSE '((TAIL NONE) (PP NONE) (ADJ NONE) (NUMBER PLUR)))


 G == the whole grammar
(SETQ GA 'ALT)


The appropriate alternative is chosen based on  CAT in the input;
 UAVS-LS is now called recursively with  this category
** In uavs-ls:

(SETQ I '((LEX GRANULE) (CAT NOUN)))

(SETQ LOOSE '((TAIL NONE) (PP NONE) (ADJ NONE) (NUMBER PLUR)))


(SETQ G
      '((ALT (((LEX TELEPHONE)) ((LEX RECEIVER)) ((LEX KIND))
              ((LEX DIAPHRAGM)) ((LEX GRANULE)) ((LEX INTENSITY))
              ((LEX MOLECULE)) ((LEX PERSON)) ((LEX POLE))
              ((LEX RESISTANCE)) ((LEX TRANSMITTER)) ((LEX WAVE))))))
```

```
(SETQ GA 'ALT)

lexical level
(SETQ I '((LEX GRANULE) (CAT NOUN)))
(SETQ I-LEX '(LEX GRANULE))
(SETQ G-LEX '((LEX GRANULE)))

** In uavs-ls:
(SETQ I '((LEX GRANULE) (CAT NOUN)))
(SETQ LOOSE '((TAIL NONE) (PP NONE) (ADJ NONE) (NUMBER PLUR)))

(SETQ G '((LEX GRANULE)))

(SETQ GA 'LEX)
 Calling av-h-ls
 *** In av-h-ls

 ******* Attribute-value pair from the grammar:
(SETQ GAV '(LEX GRANULE))

 *** Calling cntrl-av-h-ls to unify the
 input with the attribute-value pair from the grammar
In part 1 of the condition statement in CNTRL-AV-H-LS:
 a pair with same PATH was  found in the input
Value part of the pair found in the input:
(SETQ IVAL 'GRANULE)

** Calling uavs-ls from cntrl-av-h-ls  to unify the 2 values
** In uavs-ls:
(SETQ I 'GRANULE)
(SETQ LOOSE '((TAIL NONE) (PP NONE) (ADJ NONE) (NUMBER PLUR)))

(SETQ G 'GRANULE)
 Out of uavs-ls
 Success: will enrich the input with
(SETQ RESULTS
      '((GRANULE ((TAIL NONE) (PP NONE) (ADJ NONE) (NUMBER PLUR)))))
(SETQ PATH 'LEX)

 returning the result of a call to uavs-many-ls
** In uavs-ls:
```

```
(SETQ I '((LEX GRANULE) (CAT NOUN)))
(SETQ LOOSE '((TAIL NONE) (PP NONE) (ADJ NONE) (NUMBER PLUR)))

(SETQ G 'NIL)
(THE DIAPHRAGM COMPRESSES THE SMALL GRANULES!.)
```

References

[Kay 79] Kay, Martin.
Functional Grammar.
In *Proceedings of the 5th meeting of the Berkeley Linguistics Society*. Berkeley Linguistics Society, 1979.

[McKeown 82] McKeown, K.
Generating Natural Language Text in Response to Questions About Database Structure.
PhD thesis, University of Pennsylvania, May, 1982.
Also a Technical report, No MS-CIS-82-05, University of Pennsylvania, 1982.