[From: Internat.conf. *Methodology & Techniques of Machine Translation*, British Computer Society, Cranfield, 13-15 February 1984]

PRODUCTION OF SENTENCES: A GENERAL ALGORITHM AND A CASE STUDY

Giovanni Adorni and Lina Massone

Department of Communication, Computer and Systems Science
University of Genoa
via all'Opera Pia 11a, 16145 Genoa, Italy

ABSTRACT

In this paper a procedure for the production of sentences is described, producing written sentences in a particular language starting from formal representations of their meaning. After a brief description of the internal representation used, the algorithm is presented, and some results and future trends are discussed.

1. INTRODUCTION

Production of sentences by computer has been approached for several years with the general goals of generating random sentences to test a grammatical theory or converting information from an internal representation into a natural language sentence. The first approach is more oriented toward theoretical linguistics than toward functional natural language processing systems. The objective of implementing a generation system of this sort is to test the descriptive adequacy of the test grammar [10,2]. The second approach is to take some internal representation of the "meaning" of the sentence and to convert it into a surface-structure form, that is into an appropriate string of words (see, for example [4,9,6,7,3,1]).

In this paper this second line has been followed and *a* procedure for the production of sentences is described producing written sentences in a particular language from formal representations of their meaning [5]. A characteristic of the meaning representations used is that they are "relatively" universal. In fact they bear no trace of the language into which they will be eventually

mapped; it is the production procedure that imposes a language specific form on the final sentence produced.
In principle, from a given meaning representation, sentences in a number of different languages can be produced, provided that the production procedure for each language is available.
The meaning representations we are referring to in the paper do not involve any analysis of the meaning of the words. However, their format is compatible with lexical decomposition, and the procedure that we will describe accepts, without need for modification, lexically decomposed meaning representations.

In the next section the internal representation used is briefly outlined and the structure of the vocabulary is described; in section 3 the algorithm is presented and, finally, in section 4 some results and future trends are discussed.

2. INTERNAL REPRESENTATION AND VOCABULARY

The procedure described in this paper produces written sentences in a particular language starting from formal representations of their meaning (production procedure: PP). The PP accepts as its input meaning representations like, for example, the one shown in fig. 1, which produces, as output, the english sentence:

(THE CHILD SAW THE APPLE)

Formally, a meaning representation (MR) is a list of units (semantic units) made up of a predicate (in the logical sense), one or more arguments and a label. Arguments are represented with Xs or Cs followed by a code number. Xs or Cs with the same code number refer to the same unit. Labels are represented with Cs followed by a code number, and they precede the "declaration" of the unit. Labels are useful in recursive units, i.e. units that take another unit as their argument. Examples of recursive units in the MR of fig. 1 are C5 and C6.

This kind of MR representations is produced by a sentence comprehension procedure [8]which reads natural language sentences. In order to translate (lexicalise) a MR into words the procedure makes use of a vocabulary. The vocabulary is a list of "lexical entries"; each lexical entry is made up of a name and a meaning. The meaning is a list of units identical to the units of the MR except that literal codes replace number codes. Literal codes are introduced in the vocabulary to allow a general representation of the entries; these literal codes are then associated to number codes when a specific sentence is considered. The entries of the vocabulary do not correspond to "whole words" but to a set of abstract symbols: the whole words can then be obtained suitably combining the corresponding symbols.

Consider, as an example, the past of the verb "to wash", "washed", which can be represented as:

```
( (CA (P_WASH XA XB))
  (CB (MAIN CA))
  (CC (PAST CA))
    )
```

And the past of the verb "to go", "went", which can be represented as:

```
( (CA (P_GO XA))
  (CB (MAIN CA))
  (CC (PAST CA))
  )
```

From the point of view of meanings, the two verbs have the same structure, but even if intuitively "washed" can be considered as a root "wash" plus a suffix "-ed", this is not true in the case of "went". In this case we can consider the whole word "went" as the merging between an abstract root and the abstract suffix of the past tense, exactly like "washed". Therefore the vocabulary is:

```
1. ( (CA (P_GO XA))
     (CB (MAIN CA))
   )

2. ( (CA (P_WASH XA XB))
     (CB (MAIN CA))
   )

3. ( (CA (PAST CB)) )
```

The merging between 1. and 3. gives the whole word "went" and the merging between 2. and 3. gives the whole word "washed". If we assume, on the contrary, the presence of not regular words inside the vocabulary, then, starting from the MR:

```
((C1    (P_GO    X1))
 (C2     (MAIN     CD)
 (C3     (PAST     CD)
 (C4    (PERF    C1))
 (C5   (P_MARY   X1))
     )
```

which is representative of the sentence:

(MARY HAD GONE)

the system is not able to choose between:

```
( (CA (P_GO XA))
  (CB (MAIN CA))
  (CC (PAST CA))
  )
```

and

```
( (CA (P_GO XA))
  (CB (MAIN CA))
  (CC (PERF CA))
  )
```

With our choice (totally morphological vocabulary) the vocabulary contains 1., 3. and:

4. (CA (PERF CB))

In this way 1. is firstly selected; then 3. is selected and the whole word "gone" is generated; finally, 4. is analysed: with the root of the auxiliary "have-" produced during the process (see next section), gives the whole word "had". The same kind of problems are present also when dealing with irregular nouns like, for example, "child".

The list of units of a lexical entry can be divided into semantic units (COGNI) and control units (FEAT). The semantic units are representative of the meaning of a word; the control units are not representative of the meaning but they take part in the production of the whole word starting from the MR. Some entries lack the semantic units, some other the control units. All the semantic and control units are collected respectively in the cogni-list and in the feat-list; each lexical entry refers to the elements of these lists through pointers. The reason of the distinction between semantic and control units is due to the fact that semantic units are referring to the meaning of a lexical entry which is independent of the language considered; the control units are, on the contrary, specific of a particular language and allow the production of the sentences of that language. A lexical entry can contain a "lexicalization" request and some tests. As an example, the following lexical entry is now considered:

```
(NICE ((MAIN CA)
       (COGNI (CA (P_NICE XA)))
       (FEAT  (CB (XA MARK NOM))))
       (LESS  (MARK BE))
)     )
```

The LESS request activates the lexicalization procedure which produces, in this case, the auxiliary "be"; in fact this lexical entry is used to produce sentences like:

Bill is nice
Bill has been nice

. . . . .

In some cases, it is necessary to introduce a test. Consider, for example, the entry shown in Fig. 2, corresponding to the verb "to see". To choose the right units the tests have to be solved. If XA is the subject, then the following control units are selected:

(XA (MARK NOM))
(XB (MARK ACC))

If, on the contrary, the sentence is in the passive form (XB is subject), then the following control unit is chosen:

(XB (MARK NOM))

and the procedure of lexicalization is called in order to produce the whole word "by" and the suffix "-ed". An example of general structure of a lexical entry with all its specifications is given in fig. 3; fig. 4 shows an example of use of the vocabulary. In the following a phrase structure grammar driving the construction of the vocabulary is presented.

```
<VOCABULARY>      ::=  (<LEXICAL_ENTRY>*)
<LEXICAL_ENTRY>   ::=  (<ENTRY>  {(main c_node)} {<COGNI_LIST>} <AUX_PART>)
<COGNI_LIST>      ::=  (cogni <SEMANTIC_UNIT>*)
<AUX_PART>        ::=  {less <LESS_LIST>} {<FEAT_LIST>} |
                       {(test <COGNI_LIST> <FEAT_LIST> )*}
<SEMANTIC_UNIT>   ::=  (c_node   (pred <NODE>*))
<LESS_LIST>       ::=  ( {<COGNI_LIST>} {<FEAT_LIST>} )
<FEAT_LIST>       ::=  (feat <CONTROL_UNIT>*)   {less <LESS_LIST>}
<CONTROL_UNIT>    ::=  (c_node   (<NODE> (feat  value)))
<NODE>            ::=  c_node | x_node
<ENTRY>           ::=   root   |    Suffix  |   whole-word
```

3. THE ALGORITHM

The production procedure is composed of two subprocedures: the lexicalization procedure (LP) and the ordering procedure (OP). The LP, starting from a meaning representation in the form previously described, produces the unordered set of words composing the final sentence. The ordered sequence of words is then produced by the OP on the basis of the LP results and of the original MR. The task of the ordering procedure is then to assign the correct sequential order to the words in the final sentence.

1. The lexicalization procedure

The basic operations of LP are (a) identifying a particular unit in the input list (MR) (b) finding a lexical entry in the vocabulary that includes that unit as part of its meaning. The choice of the lexical entry is driven by a "principle of maximum overlap": if the vocabulary includes two or more entries having that unit as part of their meaning, LP selects the entry that shares the maximum number of units with the input list. Basically LP is a procedure specifying which lexicalizations have to be made and in which order; it is made up of three blocks of steps. The first block (steps 1-6) is concerned with the lexicalization of the sentence's predicate and subject. The second block (steps 7-8) lexicalizes any other additional argument which can be present. Finally, the third block (steps 9-10) concerns the lexicalization of adverbials, possibly included in the sentence. In the following each step is described.

- STEP 1: lexicalizing the sentence's declaration

All input lists contain one and only one unit with *a C* argument and the predicate MAIN; the C argument is called CMAIN. The first thing LP does is to identify the MAIN argument in the input-list (IL) and its "declaration", i.e.

the unit that declares the content of the MAIN argument. In the list of fig. 1 (which will be considered throughout this section) the MAIN argument is C4 and it's declaration is the unit (C4 (P_SEE X1 X2)). The corresponding lexical entry (the root verb SEE-) is then selected from the vocabulary on the basis of the semantic units. In this entry a test on the subject is present (see fig. 2), which is immediately solved to select the right control units, making use of the unit (C10 (SUBJ X1)). LP verifies also if in the control units a LESS request is present (not in our example); if so, another entry is selected on the basis of the LESS arguments. The LESS request can be recursive, that is it can appear also in the new entry; the selection of entries continues until an entry without LESS request is found. The suffix of the verb is not lexicalized at this point of the procedure because the feature units needed to select it have not yet been analysed. In fact, to choose, for example, "-s" instead of "-<Z>" (zero morpheme), LP must know that the subject is in the third singular person. For this reason, the root SEE- is stored in a temporary memory called ENTRY_MEMORY (EM) together with the corresponding node C4, waiting for the right suffix to be selected. The arguments of the sentence's declaration are stored in a list called ARG_LIST (AL) and C4,C5,C10 units are deleted from the input list since they are not needed any more. At the end of step 1 the situation is as follows:

```
IL: ((Cl (P_CHILD X1))   AL: (X1 X2)
     (C2 (ONE X1))
     (C3 (DEF X1))
     (C6 (PAST C4))     EM: ((C4 SEE-))
     (C7 (P_APPLE X2))
     (C8 (MANY X2))
     (C9 (DEF X2))
    )
```

- STEP 2: lexicalizing the subject

All the units referring to the subject of the sentence are selected from the input list; in our example they are (a) (C1 (P_CHILD X1)), (b) (C2 (ONE X1)), (c) (C3 (DEF X1)). One or more lexical entries are extracted from the vocabulary using the maximum overlap principle: firstly LP looks for an entry containing all three semantic units. If it does not find it (and this is what happens in our example), it looks for an entry presenting two of them. Since no such entry is present in the vocabulary three different entries are selected, one for each unit. The entry corresponding to unit (a) is CHILD-, which is a root; it is then added to EM. For unit (b), "-CD" is selected: it is also added to EM since it is a suffix. Unit (c) corresponds to the entry THE, which is a full voice; it is then stored elsewhere, in a special list called WORD_MEMORY (WM).

```
EM: ((C4 SEE-)(C1 CHILD-) (C2 -Ø))
```

WM: ((C3 THE))

The suffix "-Ø" is then associated to the root CHILD-: (C1 CHILD-) and (C2 -Ø) are deleted from EM and a new element (Cl CHILD- -©) is added to WM. Moreover, the argument X1 is deleted from the ARG_LIST and units C1.C2.C3 are removed from the input list. Here is the situation at the end of step 2:

```
IL: ((C6 (PAST C4))        EM: ((C4 SEE-))
     (C7 (P_APPLE X2))     WM: ((C3 THE)(C1 CHILD- -Ø))
    (C8 (MANY X2))       AL: (X2)
    (C9 (DEF X2))
    )
```

- STEP 3: lexicalizing PROGR, if present

For some sentences a PROGR ("progressive") predicate is present in a unit of the input list. Examples of such sentences are: the child is laughing, the child is being nice, the child will be studying and so on. In all these cases, LP lexicalizes that unit at step 3, producing in EM the suffix "-ing". The suffix is associated in WM either to the root verb lexicalized at step 1 (e.g. the child is laughing) or to the root verb "be-" produced by a LESS request (e.g. the child is being nice).

- STEP 4: lexicalizing PERF, if present

This step is devoted to the lexicalization of the PERF predicate, if the input list contains it. Examples: the child has laughed, the child will have laughed, the child has been laughing, the child has been nice and so on. LP produces in EM the suffix "-ed" which is associated in WM either to the root verb lexicalized at step 1 (e.g. the child has laughed) or to the root verb "be-" produced by a LESS request (e.g. the child has been nice).

- STEP 5: lexicalizing FT, if present

All sentences include necessarily a unit with one of the following predicates: PRES ("present"), PAST ("past"), FUT ("future"). These three predicates are collectively called FT ("finite tense"). Lexicalization of PRES and PAST predicates produces in EM respectively *a* suffix "-s" or "-0" and "-ed" which is associated in WM to a root according to the rule already used for steps 3 and 4. In our example a PAST predicate is present, a suffix "-ed" is generated and added to EM.

EM: ((C4 SEE-)(C4 -ED))

It is then associated to the root verb SEE- and stored in WM.

WM: ((C3 THE)(C1 CHILD- -Ø)(C4 SEE- -ED))

The PAST unit is then removed from the input list.

```
IL: ( (C7 (P_APPLE X2))
      (C8 (MANY X2))
```

```
  (C9 (DEF X2))
)
```

A FUT predicate is lexicalized with the whole word WILL (or SHALL). This is what happens in sentences like "the child will laugh". The problem is that, in this case, the root verb LAUGH- lexicalized at step 1, is still at this point lacking a suffix. This drawback is avoided noticing that the entry WILL (or SHALL) generates an extra feature unit with predicate NFT ("non finite tense") which is analysed by the next step of the procedure.

-STEP 6: lexicalizing NFT, if present
The NFT unit is a feature unit and it is therefore never originally present in the input list: it comes out as the result of the lexicalization of WILL (or SHALL). Lexicalizing a NFT unit means generating in EM a suffix "-Ø" which is added to the root verb lexicalized at step *I* if EM contains a root verb without suffix (e.g. the child will laugh); otherwise it is associated to root verbs "be-" or "have-" generated by a LESS request (e.g. the child will be laughing). At this point the first block of LP, which lexicalizes the verb and the subject of the sentence, is completed. The second block starts, which is concerned with the other arguments of the sentence's declaration, if there are any.
- STEP 7: lexicalizing additional arguments, if present
LP checks the content of ARG_LIST. If ARG_LIST is not empty, it means that additional arguments are present in the input list. In our example X2 is still in the ARG_LIST: as a consequence step 2 is repeated for X2; all units referring to X2 are now considered, generating the root APPLE-, the suffix -S and the whole word THE.

```
    EM: ((C7 APPLE-)(C8 -S))
    WM: ((C3 THE)(C1 CHILD- -Ø))(C4 SEE- -ED)(C9 THE))
Association of APPLE- with -S gives:
    EM: NIL
    WM: ((C3 THE)(C1 CHILD- -Ø)(C4 SEE- -ED)(C9 THE)(C7 APPLE- -S))
```

This step is recursive since the declaration unit can contain whatever number of arguments. At every recursion the analysed argument is removed from the ARG_LIST and the lexicalized units are removed from the input list. The recursion stops when ARG_LIST is empty. At the end of this step we have:

```
    AL: NIL
    EM: NIL
    IL: NIL
    WM: ((C1 CHILD- -Ø)(C3 THE)(C4 SEE- -ED)(C7 APPLE- -S)(C9 THE))
```

Since the input list is now empty, the LP for the sentence of fig. 1 is

completed.

- STEP 8: lexicalizing additional arguments of previous units, if present

It may happen that one unit containing one argument of the sentence's declaration includes a further argument. In this case step 8 of LP lexicalizes, in whatever order, all the units referring to this argument. Examples: the child saw Mary's father, the linguists referred to the end of the sentence, and so on. Step 8 is, of course, recursive for the following two reasons: (1) there may be more than on further argument (e.g. Mary's love for music) (2) new units being lexicalized may have other arguments in their turn (e.g. Mary's father's brother). Step 8 concludes the second block of LP. The third block is concerned with the lexicalization of adverbials.

- STEP 9: lexicalizing adverbials

An adverbial unit is a unit that has CMAIN as argument and has not already been lexicalized during first block's steps. The list of fig. 5, giving the sentence "the child is sleeping deeply", contains the adverbial unit (C6 (DEEP C2)), which is lexicalized with the entry DEEPLY. Step 9 is recursive since an input list can contain more adverbial units, as in the sentence "the child is sleeping deeply today".

- STEP 10: lexicalizing additional arguments of adverbial units, if present

An adverbial unit may have additional arguments besides CMAIN. Consider, for example, the unit (C6 (IN C2 X2)) of the list shown in fig. 6. Step 10 is concerned with the lexicalization of the units in the list having these additional arguments. These units, in their turn, may contain other arguments (as in "the child sleeps with the mother's brother"); in such cases something similar to step 8 is executed at this point.

The result of LP is the WORD MEMORY in which every element can be either a whole word (e.g. (C3 THE)), or a pair root-suffix (e.g. (C4 SEE- -ED). A special step of LP transforms every pair root-suffix into a whole word according to a set of rules, typical of the language considered. In our example:

((C1 CHILD- -Ø)(C3 THE)(C4 SEE- -ED)(C7 APPLE- -S)(C9 THE))

becomes

((C1 CHILD)(C3 THE)(C4 SAW)(C7 APPLES)(C9 THE))

2. The ordering procedure

The task of the ordering procedure is to order the words generated by the lexicalization procedure; at this purpose, also the original meaning representation is needed. OP is made up of two parts. The first part generates a fixed word order for each sentence type of the target language. This assumes that all languages have an intrinsic word order, independent of the fact that in some languages (e.g. English) sentences have a rather fixed word order, while in others (e.g. Italian) they have a more variable one. In this paper only this first part is considered, but OP has a second part which, starting from the

fixed word order, gives as output the actual (in some languages variable) word order of the generated sentence. Whereas LP may be assumed to be "quasi-universal", OP is language-specific or, at least, language-type-specific. In the following the ordering procedure for a specific language, i.e. English, is described. Basically, the OP examines the units in the MR in a certain order and, for each unit examined, it transfers the word linked to it from WM to a final workspace AC ("actual sentence"). The first move of OP is to find the CMAIN declaration in the sentence; OP identifies the argument of this unit that has a NOM control unit. If the input list includes a unit with this argument and DEF or UNDEF as predicate, OP moves from WM to AC the word linked to that unit, which becomes the first word of the actual sentence. In our example:

WM: ((C1 CHILD)(C4 SAW)(C7 APPLES)(C9 THE))
AC: (THE)

Then the other words referring to the argument are moved in AC.

WM: ((C4 SAW)(C7 APPLES)(C9 THE))
AC: (THE CHILD)

Then OP looks in input list for the FT predicate and moves in AC the corresponding word.

WM: ((C7 APPLES) (C9 THE))
AC: (THE CHILD SAW)

The same operation is then performed for the predicates PERF and NFT. Then OP produces the CMAIN word if it has not yet been produced. This is the case, for example, of sentences like: the girl will laugh, the girl is nice ..... The next step is devoted to the other arguments of the sentence's declaration, starting with the argument having a ACC unit in its lexical entry (X2 in our example). The article is generated first and then the noun.

WM: NIL
AC: (THE CHILD SAW THE APPLES)

Then the other declaration arguments are considered and, finally, the adverbial units.

4. RESULTS AND CONCLUDING REMARKS

In the previous sections a general algorithm for production of one-clause sentences has been presented. The algorithm is actually implemented in Franz Lisp on a VAX 11/750 and it is able to produce english sentences. An extension of the algorithm to multi-clause sentences is in progress. A multi-clause sentence is composed of a list of units including two or more CMAINs, each with

its own declaration. The LP for multi-clause sentences must first identify the main CMAIN and apply to it the LP for on-clause sentences. During this application, the LP will encounter another CMAIN (a subordinate CMAIN). The procedure for the main CMAIN is then immediately interrupted and the LP for one-clause sentences starts all over again with reference to the subordinate CMAIN; when this task is completed, LP resumes the procedure for the main CMAIN at the point of interruption and brings it to completion. The same "interrupt-resume" mechanism can be used to extend to multi-clause sentences the ordering procedure.

ACKNOWLEDGEMENTS

Authors wish to thank Domenico Parisi and Alessandra Giorgi for their helpful comments and discussion, leading to the implementation of the algorithm.

REFERENCES

1. G. Adorni and M. Di Manzo, "PRISE: PRoduction of Italian SEntences", Proc. AICA Annual Conference , pp. 389-400 (Naples, September 1983).

2. J. Friedman, "Directed Random Generation of Sentences", Communications of the ACM Vol. 12,6 pp. 40-46 (1969).

3. N.M. Goldman, "Conceptual Generation of Natural Language from a Deep Conceptual Base", Ph.D. Thesis, Stanford Artificial Intelligence Laboratory, Stanford University (1974).

4. S. Klein, "Automatic Paraphrasing in Essay Format", Mechanical Translation Vol. 8,3 pp. 68-83 (1965).

5. D. Parisi and C. Castelfranchi, "A procedure for sentence comprehension", Int.Rep., Istituto di Psicologia, CNR (Rome, 1981).

6. M. R. Quillian, "Semantic memory", in Semantic information processing, ed. M. Minsky.MIT Press, Cambridge, Mass. (1968).

7. S.C. Shapiro, "Generalized Augmented Transition Network Grammars for Generation from Semantic Networks", Proc. 17th. Meeting of the ACL, pp. 25-29 (La Jolla, August 1979).

8. O. Stock, "Parsing on WEDNESDAY: a distributed linguistic knowledge approach for flexible word order language", Int.Rep.312, Istituto di Psicologia, CNR (Rome, 1982).

9. T. Winograd, Understanding Natural Language. Academic Press (1972).

10. V.H.A. Yngve, "Random Generation of English Sentences", The 1961 Conf. on Machine Translation of Languages and Applied Language Analysis, pp. 66-80 (1962).

```
((C1 (P_CHILD X1))
 (C2 (ONE X1))
 (C3 (DEF X1))
 (C4 (P_SEE X1 X2))
 (C5 (MAIN C4))
 (C6 (PAST C4))
 (C7 (P_APPLE X2))
 (C8 (MANY X2))
 (C9 (DEF X2))
 (C10 (SUBJ X1))
```

Fig. 1

```
(SEE- ((MAIN CA)
       (COGNI (CA(P_SEE XA XB)))
       (TEST (SUBJ XA)
         ((FEAT ((XA (MARK NOM)
                 (XB (MARK ACC)))))
       (TEST (SUBJ XB)
         ((FEAT (XB (MARK NOM)))
          (LESS (COGNI (PASS CA))
                (FEAT (XA (MARK BY))))))))
```

Fig. 2

```
(lex-voice (T(ACCEPT(MAIN CA)
                    (COGNI (§ XA XB))
                    (TEST (§ XA)
                          ((FEAT ($ XA)
                                 ($ XB))
                    (TEST (§ XB)
                          ((FEAT ($ XB))
                           (LESS (COGNI (§ CB))
                                 (FEAT ($ XA)))))
§ ----> pointer to cogni-list
$ ----> pointer to feat-list
```

Fig. 3

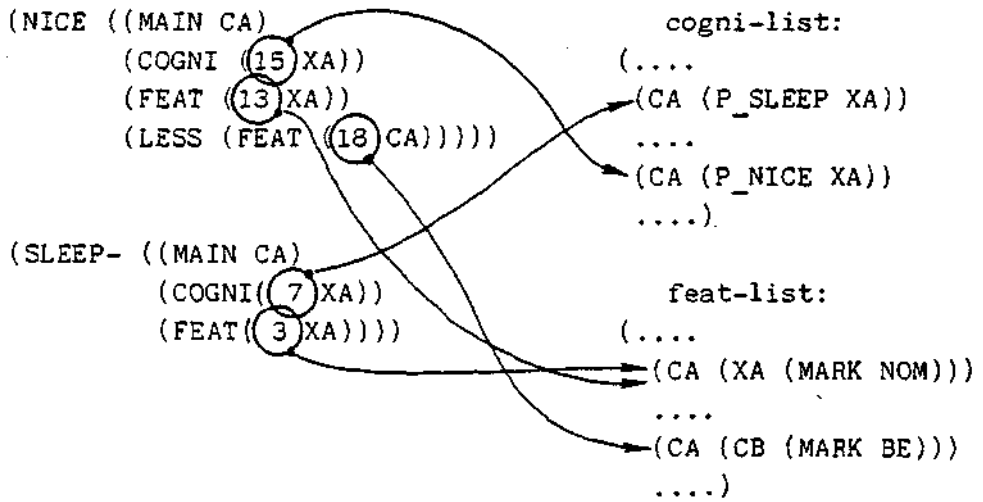

Fig. 4

```
((C1 (P_CHILD X1))
 (C2 (P_SLEEP X1))
 (C3 (MAIN C2))
 (C4 (PRES C2))
 (C5 (PROGR C2))
 (C6 (DEEP C2)))
```

Fig. 5

```
((C1 (P_CHILD X1))
 (C2 (P_SLEEP X1))
 (C3 (MAIN C2))
 (C4 (PRES C2))
 (C5 (PROGR C2))
 (C6 (IN C2 X2))
 (C7 (BEDROOM X2)))
```

Fig. 6