ET-10/51

Extracting Syntactic-Semantic Information from Cobuild Definitions and Representing it in TFS

N. Calzolari[1], S. Federici[1], C. Peters[2],
S. Montemagni[1], A. Spanu[1]

[1]Istituto di Linguistica Computazionale, CNR, Pisa
[2]Istituto di Elaborazione dell'Informazione, CNR, Pisa

CONSORZIO PISA RICERCHE, Pisa

ILC Internal Report
Pisa, August 1993

Contents

1 Introduction

In this part of the Deliverable, we present (i) the work in Pisa on the design and development of a specialised parser to extract syntactic-semantic information from the definitions contained in the Cobuild Student's Dictionary and (ii) a preliminary proposal for the mapping of such information onto a TFS representation system. The work discussed here concentrates on the analysis and extraction of information from the left hand side (LHS) of the definition and the parser must be considered as a first prototype version, to be further refined and extended as our work progresses. Similarly, once we have been able to assess the first results of the parser, it will be possible to define the most appropriate type system to represent the information extracted from this side of the definition.

Our decision to focus our efforts on the LHS has been primarily motivated by the particular style of the Cobuild definition which clearly differentiates it from those of other dictionaries. As is known, in the Cobuild dictionaries, each definition statement is the result of a careful analysis by the lexicographers of all the contexts they had available for that word sense, extracted from a corpus of more than twenty million words of running text of contemporary spoken and written English. From their studies of the corpus, it was clear to the Cobuild lexicographer that, in real language usage, it is context that disambiguates sense. Therefore, in order to represent meaning faithfully, the importance of context had to be represented, in some way, in the definition. Furthermore, as the analysis of the corpus continued, it became increasingly evident that not only were particular structural patterns frequently associated with particular senses but that very often corpus data showed not just a typical syntax but a typical pattern of lexical collocation as well (see Sinclair 1987). It was thus decided that, whenever possible, the Cobuild definition statements should not only explain the headword in each of its identified senses (as in other dictionaries) but should place it in its most typical context, as revealed by corpus evidence. Thus, while in traditional lexicography, statements are made about what words mean but very little about their use, the innovative form of the Cobuild definition equation gives attention not only to explaining the meaning of the headword (in the RHS) but also to illustrating its use by presenting the headword in its typical context (the LHS).

In order to do this effectively, a number of explanatory strategies were studied. The intention was to write the definition in a language which was as close as possible to natural language in order not just to help the reader to decode a text but to provide him/her with useful models for encoding. To do this, whenever possible, the user was to be provided with the typical grammar structures and typical collocates for each sense of a lexical entry. Compare,

for example, the first sense of the verb "to kill" as defined in the Cobuild Student's dictionary: "When someone or something kills a person, animal or plant, they cause the person, animal or plant to die" with the primary sense of the same verb in OALDCE, defined by "To put to death; cause the death of" or in LDOCE "to cause to die" (examples taken from Hanks, 1987).

All three entries give the basic meaning (to cause death) but, from the Cobuild definition, it can also be easily inferred that this sense of the verb requires an animate direct object, whereas the subject can be either human or inanimate. In fact, different senses of a word can frequently be distinguished by the different kinds of arguments or collocates associated with them. Cobuild attempts to make this clear. For example, the two senses of adore in the Student's dictionary are explained as follows: 1. If you adore someone, you love and admire them; 2. If you adore something, you like it very much. These senses are differentiated by the fact that while, in both cases, the typical subject is human, in the first the required object is also human whereas in the second it is not. Such strategies give a very good idea of how a word should be used. Other important information is also intentionally implied in these definitions by the use of "when" or "if" as initial operators, and the choice of "you" or "someone" as indicators of human arguments.

The LHS side of the Cobuild definition thus contains certain information that cannot be found in traditional dictionaries and this information is usually encoded in a consistent, coherent way. It is the linking of these elements, i.e. (i) the statement of meaning, (ii) the syntactic environment, (iii) the selectional preferences on arguments, (iv) the lexical collocational context, and (v) information on the user perspective of the verb, which, on the one hand, is unique to Cobuild and, on the other, is considered of primary importance for many problems in NLP. We therefore decided to attempt to identify explicitly this type of implicit knowledge as it gives important cues for the identification of arguments and/or modifiers of lexical items with the most appropriate features. We think that the extraction and represention of such information is of particular relevance for the construction of computational lexicons for NLP applications because it should be possible to use it in a number of tasks or subtasks, such as parsing, word-sense disambiguation, subject/object assignment, PP attachment, etc. For this reason, suitably formalized, this kind of information should be very useful for systems for automatic analysis and generation of language.

Our aim, therefore, in this first analysis has been to verify to what extent useful NLP-exploitable information could be extracted, on a large-scale, from this first part of the Cobuild definition statement. This investigation can be differentiated from previous work on the extraction of semantic information

from machine readable dictionaries, which has concentrated on classifying the headword in terms of the particular semantic features that can be derived from the genus term and differentiae. Although such investigations have gone into considerable depth, they have tended to examine subsets of definitions rather than the whole dictionary (see, for example, the work of the ACQUILEX project on words belonging to the "food" domain and on other linguistically interesting semantic subsets). On the contrary, as the methodology that we have devised to treat the LHS relies mainly on formal aspects of the definition, it is our intention in this project to apply it extensively, throughout the dictionary.

We began our analysis by studying the definitions for verbs. This decision was taken as, after a first analysis of the data, we felt that perhaps the most interesting information extractable from the LHS regarded this word-class. We were also influenced by the fact that, as far as we know, most of the work in deriving lexico-semantic information from machine readable dictionaries has so far focussed on nouns. However, we have also studied the structure of the noun definitions to see to what extent similar strategies can be successful in extracting contextual information from their LHS, and we are now beginning to study the structure of the definitions for other word classes.

In addition to the extraction of information on selectional restrictions or preferences, we are also extracting information on the superordinates or synonyms of the headwords (contained in the RHS), in particular for nouns. We feel it could be interesting to investigate how links could be created between this genus information for noun definitions and the semantic preferences identified for the different arguments of the verbs. We feel that, at the end of the project, the combination of semantic and certain syntactic data that we are extracting from the Cobuild definitions for different POSs will be particularly suitable for exploitation in the development of lexical components of NLP systems.

The rest of this section is structured as follows. In the first part, we describe our analysis of the data, the type of syntactic-semantic information we have decided to extract from the definition, and the development of our parser. The information extracted was mapped onto an Intermediate Template. Our motivations for adopting this intermediate structure are discussed in detail in a previous report (ET-10/51: Deliverable 3, p.17). The main reason is that it may well not be possible to represent everything that can be usefully extracted from the definition in the TFS formalism nor to implement it in ALEP (the problems of using ALEP to represent lexical knowledge have already been pointed out in Montemagni, 1992). The Intermediate Template gives us the opportunity both to evaluate our first results before attempting

to further formalise them, and also to store all the information extracted in a computationally tractable way, so that information which is not immediately representable in TFS or in ALEP can be kept for further analysis and possible future exploitation. Examples of the output of this stage is given in an Appendix 3.

The last part of the section describes the transferring of the information from the intermediate template into a TFS formalism and gives some examples. The next stage will be to evaluate how (and to what extent) this information can then be mapped into the ALEP system.

2 Analysing the LHS of Cobuild Definitions

Our analysis of this part of the definition was based in the first place on the Cobuild literature, i.e. on the description of the intentions of the Cobuild lexicographers and of the structure of the definitions (see, Moon 1987; Hanks 1987; Sinclair 1987, 1991).

We thus made a series of initial hypotheses. The next stage was to consult the actual dictionary data to see whether it corroborated these hypotheses. As in this stage we did not yet have any syntactically parsed input from Birmingham, we found it very useful to be able to consult the electronic version of the Student's Dictionary provided to us by Birmingham. We loaded this data under the lexical database system that has been developed and implemented at the ILC by Eugenio Picchi (see Marinai et al. 1990, Picchi 1991, for a description of how this system can be used to manage and query structured data such as dictionary archives). The flexibility of this system made it possible to formulate complex queries in order to test to what extent certain assumptions were verified by the data.

Once we had adequately tested our first hypotheses against the data, we formulated a series of templates on which we could map the results we expect to obtain for each word class. In this section, we will present the templates for nouns, verbs, and adjectives, describing the types of information that we extract and the strategies used for extraction. We will begin by discussing the work regarding verb definitions.

2.1 Verbs

As stated in the Introduction, we began by studying verbs as it appeared that, from the point of view of the kind of information we wanted to extract, the definitions for this word class contained the most interesting data.

To a large extent, the information which we can derive for verbs regards the subject, object, complement and preposition preferences of each word sense. We follow here the Cobuild philosophy, as stated by Hanks (1987), which is that "selection preference" for arguments is a far more appropriate concept than "selection restriction". We attempt to classify the selected arguments in terms of the features which they should bear and/or their typical semantic domain. The information we extract is not only semantic but also syntactic, or better, a combination of the two. feel that if our results are to be useful as input for other systems we must aim at producing entries which are as complete as possible (see an earlier project report by Montemagni et al. 1992, in which a first proposal for the representation of data extracted from Cobuild definitions also considered orthographic, phonetic, and syntactic data).

Other data that we are deriving from the LHS should help us to classify the verb type on the basis of certain inferences that can be made from the particular formulation of the definition. For example, we are particularly interested in testing whether distinctions of the if/when, you/someone type in the dictionary were actually reliable and consistent in practice.

According to the Cobuild literature (for example, Sinclair 1991), the hinge "when" is adopted for animate subjects when the action of the verb can be considered as an inherent action or typical activity of the subject (e.g. the verb "reign" is defined by "When a king or queen reigns..." and "sneeze" by "When you sneeze...") or for inanimate objects when the action appears to be inherent in the nature of the object (e.g. "When the sun rises..."). "If", on the other hand, is used for actions which involve some kind of choice or are not inherent (e.g. "If a king or queen abdicates...", "If a meeting or trial adjourns..."). The "you/someone" alternation in the subject position is used to distinguish between human actions that are considered neutral and those that are not. The implication of "you" is that the sentence expresses something that anyone might reasonably and normally do while "someone" is reserved for negative or unlikely actions. Other words used by Cobuild to imply particular features for the verb arguments are "somebody", "person", "people", "something", "things", etc. We have attempted to codify the information which can thus be inferred from the use of these particular definition strategies in order to be able to assess to what extent they can be considered reliable and/or useful knowledge for an NLP lexicon.

The information which we are currently deriving for verbs is shown in the following template and discussed in detail below[1].

[1] '*' indicates 0 or more occurrences, '()' indicates optionality.

```
TEMPLATE FOR VERBS

GENERAL_INFO:
     DEF_No.:
     SENSE_No.:
     DEF_TYPE:
     LEMMA:
     ENTRY:
     GENUS_INFO:
     INFLECTION:
     GRAM:
     (SEC_GRAM:)*

PREFERENCE_INFO:
      VOICE:
     (INFERENCE:)
     (PHRASE_TYPE:)
     (SUBJ_INFO:)
     (OBJ_INFO:)
     (OBJ2_INFO:)
     (OBLIQUE_INFO:)
     (CLAUSE_INFO:)
     (ADJUNCT_INFO:)
     (TO-INF_INFO:)
```

Each of the INFO type attributes can be further expanded as shown in the examples below:

```
GENUS_INFO:{PROV_SUPERORDINATE:valueV PROV_SYNONYM:value}

SUBJ_INFO:({SPECIFIC:value V TYPICAL:value V NULL})
          (SUBJ_FEATURES:features)
          (SUBJ_PREMOD:values)
          (SUBJ_POSTMOD:values)

OBJ_INFO:({SPECIFIC:value V TYPICAL:value V NULL})
         (OBJ_FEATURES:features)
         (OBJ_PREMOD:values)
         (OBJ_POSTMOD:values)
```

```
OBLIQUE_INFO:({OBLIG_PREP: prep V PREF_PREP: prep })
             ({SPECIFIC:value V TYPICAL:value V NULL})
             (OBLIQUE_FEATURES:features)
             (OBLIQUE_PREMOD:values)
             (OBLIQUE_POSTMOD:values)
```

It can be seen from the template that we are extracting two types of data. In the first block of General Information, we mainly have data that is already given explicitly in the syntactically parsed definitions received from Birmingham. We simply extract this and store it directly for future use[2].

The second block is the set of information that is contained only implicitly in the LHS of the definition and which we derive by our analysis. This block consists of information which classifies the verb with respect to the preferred features of its different arguments, to its preferred form, and gives a value to the type of action represented in the given sense. In the following, we will give details and examples of the different attributes in the template and their possible values, and briefly illustrate the types of procedures used to extract the information and map it onto the template.

2.1.1 Attributes and Values

GENERAL INFORMATION

As stated, this first block of Attributes mainly regards values which are extracted more or less directly from the Birmingham input and which are necessary to us for our further treatment and management of the data. Apart from SEC_GRAM, all these Attributes are obligatory.

DEF_No.: Extracted directly from the Birmingham input. Used to univocally identify each definition.

SENSE_No.: Extracted directly from the Birmingham input. Identifies the particular sense of the entry, as given in the dictionary.

DEF_TYPE: Extracted directly from the Birmingham input. Identifies the particular definition strategy used by the Cobuild lexicographers when writing the definition statement. These strategies are defined in (Allport et al. 1993a and b).

[2]We refer to the input data in the following as the Birmingham input and indicate fields from this input using single inverted commas whereas the fields in our template are referred to using capital letters. However, there is not necessarily a direct one-to-one relationship between the values of the Birmingham attributes and ours; for example, the Birmingham data gives 'lemma' which contains the inflection values, whereas our LEMMA contains the value for the base form of the lexical item being defined.

ENTRY: Extracted directly from the Birmingham input, normally from the content of 'head'. Contains the lexical item actually being defined.

LEMMA: Contains the base form of the lexical item being defined (the dictionary headword). This information is derived from the first item contained in the list under 'lemma' in the Birmingham input. The distinction between ENTRY and LEMMA is necessary as the lexical item being defined is not necessarily identical to the headword of the entry. For example, we have three definitions under the LEMMA "abide", where the values for ENTRY are "can't abide", "abide" and "abide by". The value under LEMMA permits us to relate each of these three examples back to its base lemma All the following attributes in the template refer to values of ENTRY rather than of LEMMA.

GENUS_INFO:PROV_SUPERORDINATE V PROV_SYNONYM: The values for the genus information must be considered provisional. At the moment, we are simply rewriting the Birmingham values given for this field under 'superordinate' or 'synonym' in the RHS. However, when we begin to analyse this part of the definition, it will be necessary to assess and attempt to standardise this data before beginning work on it as actually it is not homogeneous.

INFLECTION: The Cobuild dictionary gives the total inflection for the entire entry immediately after the headword rather than separately for each grammatical category. This strategy is economic with regard to space but of course relies on the users' knowledge for the correct identification of the inflection of any particular homograph of the lemma. We thus have to analyse the 'lemma' field in the Birmingham input in order to derive the correct inflection, depending on the POS being treated. For non modal verbs we currently simply rewrite the values given in this field. For nouns and adjectives, we must use a different strategy (see below).

GRAM and SEC_GRAM: The value is taken from the 'grammar' field in the Birmingham input. However, when the grammar field contains one or more "or" this indicates that there is more than one grammar possibility for this particular sense. In this case, the first possibility is written in GRAM, the second (or others) in SEC_GRAM. This distinction is needed by our procedure in order to identify the correct template for the analysis (for details, see the section describing the Parser).

PREFERENCE INFORMATION

In this second block of data, we write the values that we derive from the analysis of the LHS of definitions performed by our parser. Apart from VOICE, all of these attributes are optional.

VOICE: For each word category or subcategory, Cobuild has an unmar-

ked explanation strategy against which marked strategies can be contrasted. For transitive verbs, for example, the unmarked strategy is the active voice against which some verbs are defined using a passive construction to make their preference for this construction clear, e.g. "If someone or something is acclaimed, ...". Similarly, other verbs are defined using the progressive voice, e.g. "If someone or something is acting up, ..." This attribute thus gives a value for the preferred voice or tense used in this sense of the verb, as shown by the corpus. Possible values: Active/Passive/Progressive. The value is derived by analysing the value of 'head1' in the Birmingham input and comparing it with the contents of 'lemma' which contains inflection information.

If 'head1' contains "is" or "are" and the following string is equal to the third item in the contents of 'lemma', then VOICE: Progressive If 'head1' contains "is" or "are" and the following string is not equal to the third item in the contents of 'lemma', then VOICE: Passive. Otherwise, VOICE: Active.

INFERENCE: As we have already mentioned, one of the most interesting features of the definitions in the Cobuild dictionary is the use of "if/when" and "you/someone" in an attempt to provide the user implicitly with an idea of the "perspective" of the verb. We thought that it would be both useful and interesting to derive this kind of information which can be used to classify in some way the perspective the user has on the verb.

For definitions where Def_Type = 1, the value of Inference is derived from the value contained in field 'hinge' together with that contained in 'co-text1'.

Here below, we give some examples:

- Hinge = if
co-text1: Does not contain one of the values in List2 in Appendix2
(i.e. you, someone, people)
Value: likely
- Hinge: when
co-text1: Does not contain one of the values in List2
Value: inherent
- Hinge = if
co-text1 contains "you"
Value: possible, likely
- Hinge = if
co-text1 contains "someone"
Value: possible, negative/unlikely
- Hinge = when
co-text1 contains "you"
Value: inherent, likely
- Hinge = when
co-text1 contains "someone"

Value: inherent, negative/unlikely
- Hinge = if
co-text1 contains "group", "people"
Value: possible, collective
- Hinge = when
co-text1 contains "group", "people"
Value: inherent, collective
For definitions where Def_Type = 3, Hinge = "if" or "when", and 'Projection' contains "you say that", "you describe"
Value: Subjective

Currently, for other values of 'hinge', we make no inference.
Examples of results can be seen in Appendix 3.

PHRASE_TYPE: Indicates the kind of phrase we are treating. So far we have only treated verbal phrases. If DEF_TYPE = 1, then the value of PHRASE_TYPE is verbal. The definition is then analysed in the same way as that of a verb.

ARGUMENT PREFERENCE INFORMATION

The main type of information that we extract from the LHS of the definitions for verbs regards syntactic and semantic information on the preferences of their arguments. Examples of how this type of information is structured have been given above in the Template Let us now explain the meaning of the different attributes and the kind of values they can take.

The values for the preference data are acquired from an analysis of the data contained in 'co-text1' and 'co-text2' in the Birmingham input. The co-texts contain information on the preferred arguments of the verb, presented at different levels of generality, to give the user as clear as possible an idea of the kind of arguments required. We have identified three levels of arguments: (i) the almost obligatory argument of the "If a horse gallops..." type, (ii) the very general argument of the "If you copy something ..." type and (iii) an intermediate level in which the range of arguments is restricted to a particular area or areas, e.g. "If something such as a path or river forks...".

In the first case, the definition is stating that the preferred subject of the verb "gallop" is normally a horse; the inference is that when using this verb the subject slot must be filled by something that is definable as a horse (e.g pony, stallion, mare, Black Beauty, and so on). In our template, this kind of argument has been tagged as SPECIFIC.

In the second case, Cobuild uses a finite list of words to express various kinds of very general arguments. Depending on the particular word used, different features can be derived for the arguments of the given sense of the verb in question. The example above implies that "copy" in this sense requires a human subject but an inanimate object. From our analysis of the data, we have constructed a list of these words, see Appendix 2. Thus, when our procedure finds one of the words contained in this list, the relevant features, as given in the list, will be derived for that argument of the verb. The possible values for features are currently:

```
FEATURES: +/-Anim,+/-Hum, +/-Masc, +/-Fem, +/-Coll, +/-Animal,
+/-Plant,+/-Count
```

In this stage of extraction, the features are simply stored as above in a flat list. They are modelled in a hierarchical structure in the TFS representation.

In the third case shown above, the definition implies that the subject of the verb "fork" in this sense will be inanimate and typically belongs to the semantic domain(s) exemplified by the particular words which appear after the "such as" trigger. In this case, we derive general features, as above, for the argument from the general word used (here it is "something") and write the strings following the trigger as values of TYPICAL. The possible strings used as triggers are contained in another List (see Appendix 2).

SUBJ_INFO also contains SUBJ_PREMOD and SUBJ_POSTMOD where SUBJ_PREMOD is used for any strings preceding the identified subject and SUBJ_POSTMOD is used for any strings following it. This distinction closely models how these modifiers are syntactically realised within the definitions. We have made it in order to facilitate future processing. In fact, whereas the data which we tag as PREMOD is normally an adjective or attributive noun, frequently the data which is tagged as POSTMOD is a clause which will probably be subjected to further analysis later on.

OBJ_INFO is used for values for the object of the verb and expanded as above for SUBJ_INFO

OBJ2_INFO is used for values for second object when the verb is used with two objects. The value of 'grammar' will be VB WITH OBJ AND OBJ. The values for this Attribute are as shown above for SUBJ_INFO and OBJ_INFO.

The OBLIQUE_INFO Attribute is expanded as shown on the following page:

```
OBLIQUE_INFO:({OBLIG_PREP: prep V PREF_PREP: prep V NULL})
             ({SPECIFIC:value V TYPICAL:value})
             (OBLIQUE_FEATURES:features)
             (OBLIQUE_PREMOD:values)
             (OBLIQUE_POSTMOD:values)
```

OBLIG_PREP is used when the value of the 'grammar' field for the verb specifies the use of a preposition, e.g. the ninth sense of "break" is categorised as "VB WITH 'for"' and "for" will be the value of OBLIG_PREP.

PREF_PREP instead is used when the value of a preferred preposition for the verb is not specified in 'grammar' but can be derived from an analysis of the LHS of the definition. For example, with sense 2 of "advance", defined as "If you advance in something you are doing...", the preposition "in" is not specified in the grammar field but is derived as value of PREF_PREP.

The distinction between OBLIG_PREP and PREF_PREP is thus determined by the way in which these prepositions appear in the lexical entries in the dictionary and reflects a parallel distinction made by the lexicographer, clearly on the basis of corpus evidence. However, on a first examination of the data, it is not always evident why in one case a particular preposition has been classified explicitly as part of the grammar field whereas another, which appears to be equally "obligatory" can only be inferred from the definition. For example, compare the first two senses of "listen":

1. VB "If you listen to someone who is talking or to a sound, ..."
2. VB WITH 'for' "If you listen for a sound, ..."

Once we have processed all the definitions, it will be easy for the lexicographers to compare and evaluate this kind of data, correcting it if necessary.

2.2 Nouns

The noun definitions in the Student's dictionary are not so rich in contextual information as those for verbs. Noun collocates tend to be many and various and do not necessarily occur in any regular structural relationship with the noun itself. Unlike verbs, it is not true that lexical selection preferences are generally associated with particular syntactic structures (see Hanks, 1987). Thus frequently it was not possible for the Cobuild lexicographer to contextualise the LHS of the definition in any way. Thus we find definitions of the following type: "A bank is a place where you keep your money in an account", and "A bank is also the raised ground along the edge of a river or lake", where it is entirely the semantic content of the RHS that distinguishes between the two senses of bank.

However, on the other hand, the corpus did show that there were many cases where the combined collocational and syntactic preferences of a noun

provided a basis for the adoption of an explanation strategy to give the user implicit information on such preferences. For example, one sense of administration is defined by "The administration of a company, an institution, or a country is ..." to indicate that administration in this sense typically selects an argument introduced by the preposition "of" which is normally followed by words such as "company", "institution", "country".

We have thus used similar strategies to those developed for analysing the LHS of verb definitions to extract, where possible, information on the syntactic and collocational preferences for nouns. So far, we have only tagged this information as collocational (*COLLOC_INFO) without attempting to further define its role (as has been done for the collocational information for verbs - tagged in terms of their syntactic role). In a second stage, we intend to analyse this data in more depth.

Here below, we illustrate the template and describe the information we are currently deriving for nouns.

```
GENERAL TEMPLATE FOR NOUNS

GENERAL_INFO:
     DEF_No.:
     SENSE_No.:
     DEF_TYPE:
     GRAM:
     (SEC_GRAM:)*
     LEMMA:
     ENTRY:
     GENUS_INFO:
     INFLECTION:
     (PHRASE_TYPE:)

PREFERENCE_INFO:
     (FORM:)
     (PRECOLLOC_INFO:)
     (POSTCOLLOC_INFO:)
```

Each of the INFO type attributes can be further expanded as shown in the examples on the following page:

```
GENUS_INFO:{PROV_SUPERORDINATE:valueV PROV_SYNONYM:value}

PRECOLLOC_INFO: values
POSTCOLLOC_INFO:({OBLIG_PREP:prepV PREF_PREP:prep })
                ({SPECIFIC:value V TYPICAL:value V NULL})
                (COLLOC_FEATURES:features)
```

2.2.1 Attributes and Values

Only the information types which differ from those described for the verb template will be discussed.

INFLECTION: For uncount, mass and collective nouns, no value is read. For count nouns, the first two values of the 'lemma' field of the Birmingham input are read.

FORM: In the definition statements for count nouns, the unmarked strategy is to use the singular form with the indefinite article, e.g. "A bag is ...". However, if corpus evidence suggests that a particular noun is commonly used in the plural form then this is reflected in the definition, e.g. "Beads are ...". We tag this information explicitly by assigning the value "plural" to the FORM attribute.

*COLLOC_INFO: This tag cover two attributes: PRECOLLOC_INFO and POST_COLLOC which contain the noun collocates found, respectively, in 'co-text1' and 'co-text2' of the Birmingham input for which, so far, we do not provide a more fine-grained interpretation.

PRECOLLOC_INFO: used for any strings which precedes the entry item (found in co-text1). This attribute does not have a complex structure as its values only range over adjectives or attributive nouns, which we are unable to differentiate as our input does not provide POS tagged data. For example, sense 11 of "cover" is defined by "Bed covers are ..." In this case, the value of PRECOLLOC_INFO will be "bed".

POSTCOLLOC_INFO: used for any string or sequence of strings which follows the entry item (found in co-text2). The information is analysed in much the same way as described above for verb arguments. Again, we have identified three levels of generality for the preferred collocates: very specific collocates as in "A box in a theatre is ...", very general ones of the "An abundance of something..." type, or an intermediate level of generality where the typical area is indicated e.g. "A book of something such as stamps, matches, or tickets ..." In the same way as for verbs, in the first case we

derive "theatre" as value for SPECIFIC, in the second case we derive values for COLLOC_FEATURES from the features assigned to "something", and in the third case we derive "stamps", "matches", and "tickets" as values of TYPICAL and values for COLLOC_FEATURES from "something". The preposition is assigned to OBLIG_PREP when this information is specified in the 'grammar' field, otherwise it is assigned to PREF_PREP, similarly as for verbs. In the three cases above, the preposition "in" or "of" will be assigned as value for PREF_PREP.

Another type of noun collocate is exemplified by the following piece of definition for one sense of "bunch": "A bunch of bananas, grapes or other fruit is..." In this contextualization of "bunch" we are given the information that this sense of the word is typically used with "bananas" and "grapes" and "bananas" and "grapes" both belong to the "fruit" domain. The problem is that it would also seem possible to infer that "bunch" could also be generally collocated with "fruit" whereas, correctly, it can only be collocated with types of fruit which have the particular characteristic of bananas and grapes, i.e. they grow in bunches. We need to examine more examples of this type before deciding how this information should be treated. At the moment, "bananas", "grapes" and "fruit" are all acquired as values of specific, thus loosing the information that "bananas and grapes are fruit (important for sense matching) and apparently acquiring (incorrectly) the information that we can have a bunch of any fruit.

2.2.2 Processing Particular Nouns as Verbs

The procedure which analyses the definitions begins by examing the POS contained in the 'grammar' field of the Birmingham input and the value assigned to 'Def_type' in order to determine the correct processing strategy to be adopted. The different definition types used by Cobuild are discussed in (Allport et al. 1993a and b). In the definitions we have studied so far, noun definitions are generally assigned to Def_Type 3. However some noun definitions are of Def_Type 1, which is characterized as being of the "if/when" type and is typically used for verb definitions, e.g. "When you focus a camera you ...", "If you follow someone's instructions...". We found that, when this defining format is used for nouns, the noun defined appears in fact as argument of the verb which is head of this part of the definition. For example, the two senses of "attention" in the Student's dictionary are both classified as uncount nouns and are defined as follows:

"If you give something your attention, you look at it, listen to it, or think about it carefully"

"If something is getting attention, it is being dealt with"

whereas

"If you pay attention to something, you watch it, listen to it or take notice of it" is listed at the end of the entry for "attention", as a Phrase, whose headword is evidenced as "pay attention".

Examining these definitions we found that, in all three cases, the RHS superordinate or synonym was a verb or verbal phrase and that, although the formal headword in the first two cases was a noun, what was really being defined in every case was a verbal phrase. Thus, when we find a noun definition with Def_Type 1, we construct the value for ENTRY by taking not only the value found in 'head' in the Birmingham input but also preceding it with the contents of 'co-text1'. Thus for the first two examples above, our constructed values for ENTRY are "give something your attention" and "getting attention". At a later stage, it should not be difficult to derive the values "give attention" and "get attention" from this data. The entry is then tagged explicitly as being a verbal phrase, and we analyse it in the same way as for verbs. In the results produced by our parser for the three definitions cited above, it can be seen that, by processing the definitions in this way, the formal difference between them maintained in the dictionary disappears and instead their similarity is evidenced.

We feel that this kind of information should be interesting because it exploits another special feature of Cobuild, i.e. the attention paid by Cobuild to phrases in their defining strategy and, at the same time, provides explicit information on phrasal constructions which would be of great importance to NLP lexicons.

```
def_no              : 1498
sense_no            : 1
def_type            : 1
gram                : UNCOUNT N
lemma               : attention
entry                                    : give something
                                           your attention
genus_info          : superordinate3     : think about
                      superordinate2     : listen to
                      superordinate1     : look at
phrase_type         : verbal
voice               : active
infererence         : possible likely
subj_info           : subj_features1     : +anim, +hum
```

```
def_no              : 1499
sense_no            : 2
def_type            : 1
gram                : UNCOUNT N
lemma               : attention
entry                                   : is getting
                                          attention
genus_info          : synonym1          : is being dealt
                                          with
phrase_type         : verbal
voice               : progressive
inference           : possible
subj_info           : subj_features1    : -anim

def_no              : 1502
sense_no            : 2
def_type            : 1
lemma               : attention
entry                                   : pay attention
genus_info          : synonym3          : take notice of
                      synonym2          : listen to
                      synonym1          : watch
inflection          : attention
gram                : PHRASE
phrase_type         : verbal
voice               : active
infererence         : possible likely
subj_info           : subj_features1    : +anim, +hum
obj_info            : obj_features1     : -anim
```

It appears to us that this use of Def-type1 to define nouns occurs when we have nouns used together with so-called "support" or lexically empty verbs, see "give" and "get" in the examples above. There was an interesting discussion on this point at the project meeting in Birmingham at the end of June and we intend to further investigate this type of definition. The fact that our treatment reveals the similarity between definitions treated by the dictionary as Nouns and others classified as Phrases may be found useful by the Cobuild lexicographers who may want to re-examine their treatment of these items.

2.3 Adjectives

We are now using very similar strategies to analyse definitions for adjectives. The template adopted is shown below. However, we have only just begun to examine these definitions and may well alter and/or expand this template as our analysis becomes more exhaustive.

```
GENERAL TEMPLATE FOR ADJECTIVES

GENERAL_INFO:
     DEF_No.:
     SENSE_No.:
     DEF_TYPE:
     GRAM:
     (INFLECTION:)
     LEMMA:
     ENTRY:
     GENUS_INFO:
     (PHRASE_TYPE:)

PREFERENCE_INFO:
     (COLLOC_INFO:)
```

For which we have the following expansions of values:

```
GENUS_INFO:{PROV_SUPERORDINATE:valueV PROV_SYNONYM:value}

COLLOC_INFO:({SPECIFIC:value V TYPICAL:value})
            (COLLOC_FEATURES:features)
```

2.3.1 Attributes and Values

The attributes and their values are very similar to those described above for nouns. We are currently extracting information on the type of noun collocate which is qualified by a given adjective. In the case of adjectives too, the Cobuild definitions present different levels of generality for the preferred collocates. Thus, we have examples ranging from very specific noun collocates, e.g. "A fizzy drink is..." or "Floral cloth, paper or china has ..." to very general indications such as "Someone who is livid...", "An effusive person is ...", "If you are flabbergastered...". In the first cases, we derive values for the SPECIFIC attribute, in the second we infer values for the COLLOC_FEATURES attribute. With reference to the values for INFLECTION, if the 'lemma' field of the Birmingham input contains more than one item, then the first three values are read. Otherwise, no value is read.

3 The Parser

In the previous chapter, we have described the type of information we are deriving from the LHS of the Cobuild definitions. We will now give an idea of the procedure used to extract this information.

The first draft of the LHS parser was based on a preliminary example of output from Birmingham in which each word in the definition was accompanied by a POS tag (see our Handout presented at the meeting in Birmingham in June 1993). As this kind of tagging was not, however, considered practicable by Birmingham and thus not included in the final output of their syntactic parser, with respect to the work described in that Handout, we have had to develop different more complex pattern matching techniques to analyse the information contained in the definitions. The general strategy of the parser is to try to identify significant chunks within the defintion on the basis of clues provided by strings or sequences of strings which have been identified as typical delimiters of meaningful items of information. This strategy can perhaps be considered, in the future, as a possible alternative to traditional parsing techniques.

Here below we give just some examples of how these techniques have been developed to treat the co-text data. The programming details for this part of the procedure can be found in Appendix 1, in the section "Inside Low-level Functions", see functions "elabmatch_info", and the results for our test sample of data are given in Appendix 3.

3.1 Analysing the Co-text Data

The co-text data is processed as follows. The co-text must first be read to see if it contains "or" and ",". Each item or group of items which have been divided by "," and/or "or" is considered separately for the analysis.

The co-text is then analysed to see whether values for the SPECIFIC, TYPICAL or *_FEATURES attributes can be derived:

- If the first item contained in the co-text is contained in List2 in Appendix 2 (e.g. you, someone, something, etc.), and if the co-text contains one of the "triggers" given in List4 in Appendix 2 (e.g. such as, for example), then the string, or sequence of strings, following the "trigger" will be analyzed to extract values for the attribute TYPICAL and values for *_FEATURES will be derived from the item which is listed in List2.

- If the first item contained in the co-text is contained in List2 in Appendix 2 (e.g. you, someone, something, etc.), and if the co-text does not contain any of the "triggers" given in List4 in Appendix 2, then values for *_FEATURES

will be derived from the item which is listed in List2.

- Otherwise, if the first item contained in the co-text is not contained in List2, then the co-text will be analysed to derive values for SPECIFIC.

In the following, we will give an idea of how the co-text is processed to extract information on "SPECIFIC" arguments. In this case, each item or group of items in the co-text are read. For each group of items, the last one is presumed to be the head argument and those preceding are normally taken as its modifiers. When the modifier is attached to one of the arguments in the co-text that is not the first argument, it is taken as modifying only its adjacent head. Thus, if in our input data we have:

```
'(a)
'(word)
'(or)
'(a)
'(musical)
'(note)
```

we derive "word" and "note" as values for SPECIFIC and "musical" as the value for PREMOD of "note".

Whereas any string preceding the first argument in the co-text is taken as referring to each argument. Thus, if we have

```
'(someone's)
'(advice)
'(or)
'(suggestion)
```

we derive "advice" and "suggestion" as values for SPECIFIC, each with an attached PREMOD: someone's.

When we find cases of embedded "of" groups preceding an argument, this group is read as the value of the modifier of the head. For example, from

```
'(a)
'(piece)
'(of)
'(writing)
'(or)
'(speech)
```

we derive as SPECIFIC values: "writing" and "speech", and "piece of" as the value for the PREMOD of "writing". These classes of premodifiers may

be further analyzed to derive other semantic properties of the lexical item being defined in a second stage.

Whereas, if the "of" group follows the noun, the group is read as a postmodifier of the noun. So with

```
'(an)
'(attitude)
'(,)
'(position)
'(,)
'(or)
'(way)
'(of)
'(behaving)
```

we derive as SPECIFIC values: "attitude", "position", "way" and "of behaving" as the value for the POSTMOD of "way".

When the embedded "of" group includes one of the special items in List2 (e.g. someone, something) then it is taken as modifying each preceding argument.

Thus,

```
'(the)
'(rate)
'(or)
'(speed)
'(of)
'(something)
```

gives as SPECIFIC values: "rate", "speed", each of which has "of something" as its POSTMOD.

The co-text is more difficult to handle when it is an unbroken sequence of strings, i.e. it contains no ",", "or", embedded "of" groups and thus a first splitting is not possible. In this case, the string is acquired to see whether it contains one of the items in List2 (e.g. you, something, etc., but also including which, who, and that) but not in the first positions. If so, the string is divided at this point. For example, if we have

```
'(the)
'(food)
'(you)
'(are)
```

```
'(eating)
'(,)
```

we derive "food" as value for SPECIFIC and "you are eating" becomes a value for POSTMOD. This phrase may then be subjected to further analysis in a second stage of development of the parser.

As stated above, when the first item of the co-text contains one of the special items in List2 (e.g. someone, something) values for *_FEATURES are derived from this item. In this case, any data following this item is taken as a post modifier of it. Thus, for example,

```
'(something)
'(that)
'(you)
'(have)
'(been)
'(offered)
```

gives "-anim" as value for *_FEATURES and "that you have been offered" as value of POSTMOD.

If the string following the item in List2 contains "or" or "," then it is divided into separate groups as described above and the features derived for the List2 item are assigned to each of the post modifying values. To give an example, with the definition "If you admit to something bad, unpleasant, or embarassing"

```
'(something)
'(bad)
'(,)
'(unpleasant)
'(,)
'(or)
'(,)
'(embarassing)
```

is rewritten as "something bad", "something unpleasant", "something embarassing", and our parser produces the following output:

```
subj_info          : subj_features1     : +anim, +hum
obj_info           : obj_postmod1       : bad
........           : obj_features1      : -anim
........           : obj_postmod2       : unpleasant
........           : obj_features2      : -anim
```

```
........          : obj_postmod3      : unpleasant
........          : obj_features3     : -anim
```

3.1.1 Deriving *_FEATURE Values

As has already been stated, when the co-text data contains one of the special items listed in List2 in the first positions, then the appropriate values for *_FEATURES are read from this list.

However, when the cotext gives us values for SPECIFIC arguments then we attempt to find feature values for these arguments. For instance, feature values are found for SUBJ_INFO data using the following procedure:

(i) considering the articles included in the contents of 'cotext1' and the inflection of 'head1'. In this way, it is usually (although not always) possible to derive the value for COUNT.

a) if co-text1 contains "a", "an" or "one", then the value +COUNT is derived for each value of SPECIFIC.

b) if co-text1 contains no article (see List8) and the form of the verb in 'head1' is in the third person singular, then the value -COUNT is derived.

c) if co-text1 contains no article (see List8) and the form of the verb in 'head1' is in the third person plural, then the value +COUNT is derived.

d) Otherwise, no value for COUNT is derived.

(ii) examining the strings contained in the relevant 'match' field in the RHS.

If the content of this field is any of the strings contained in List3, the relevant values are taken and inserted in *_FEATURES

For example, for "abdicate1", we have the following piece of input from Birmingham:

```
(lhs-1
      (co-text1
        (match1
           '(a)
           '(king)
           '(or)
           '(queen)
        )
      )
```

```
        (head1
              '(abdicates)
              '(,)
        )
  )
  (rhs-2
        (match1
              '(he or she)
```

where the presence of the article "a" gives us the value of +COUNT. and where the RHS match1 "he or she" matches the LHS match1 "a king or queen". (When the RHS match contains an "or", the strings before and after the "or" are associated with the relevant strings of the LHS match, e.g. in this case we have "king" associated with "he", and "queen" with "she".) The output of the parser is as follows:

```
subj_info       : subj1             : specific: king
                : subj_features1    : +anim,+hum, +masc, +count
                : subj2             : specific: queen
                : subj_features2    : +anim,+hum, +fem, +count
```

In the following and in Appendix 1, we give a technical description of our parser. However, it must be remembered that the parser is still in a preliminary stage of development and we are now evaluating our first results. We have only tested it so far on the first output from Birmingham, approximately 150 definitions (the results are given in Appendix 3), clearly an insufficient number to cover all possible significant patterns to be found in the definitions throughout the dictionary. We thus expect to have to modify the parser and extend its core once we have the possibility to test over a larger sample of definitions.

3.2 The Implementation

To implement the Pisa Syntactic-Semantic Parser, we have developed a subset of functions so that the program, written in the C programming language, is as self-explanatory as possible. For the sake of clarity, this set of functions can be subdivided as follows:

- list manipulation functions
- string manipulation functions
- grammar functions
- input/output structure manipulation functions

where lists are implemented as character strings operating on blocks of consecutive characters delimited by spaces ('words'), and i/o structures are binary tree-like structures whose node are accessible by path specification.

All the functions are described by means of their argument types (between brackets) and any result type (following the left bracket).

To implement the Parser, we used the C programming language on a Sun3/50. The parser is a 59Kbyte source code and a 88Kbyte executable file. The input file (the output from Birmingham) is read from the "birm.inp" file and the output is written in the "sempars.out" file. A technical description of the Parser is given in Appendix 1.

4 Mapping the Results onto the TFS Representation System

As described in Deliverable 3 and stated above, we decided to store all the information extractable from the definitions on an Intermediate Template. The reason for this is that, given the already documented limitations of the ALEP system as far as the treatment of lexical knowledge is concerned, we felt that it was important to find a way to be able to store all the information we extracted, even if not everything could be directly exploited by ALEP in its present stage of development or within the immediate context of the project.

The next step is thus to translate as much as possible of the information extracted from the LHS of the definition and stored in this Intermidiate Template onto the TFS representation structure, first described in Montemagni et al, 1991, and now being revised and developed. In fact, up until March 1993, with respect to the TFS our attention was focussed on representing the information from the RHS. The decision to concentrate our efforts for now on the extraction of syntatic-semantic information from the LHS has meant that we have now had to study a development of the TFS representation to incorporate the type of information which is currently being derived by our parser.

We have thus now begun to study the design of a Type System which can model the information extracted from the LHS with the aim of defining a preliminary prototype representation. This will be experimented and revised in the next stages of the project.

What we are presenting here refers to the type system presented in previous deliverables, which has been modified in order to encode the information extracted from the LHS. Therefore, in this section, we present only those integrations and changes which have been made with respect to the complete version presented in Deliverable 2. Most of these changes have been moti-

vated by the necessity to formalize the new information extracted from the LHS; however, some have been introduced in order to adjust the type system to the common interface agreed with the partners on the one hand, and to the requirements of HPSG theory on the other. These last two points are strictly connected, but we prefer to differentiate between them because of the presence of Cobuild-specific attributes in the common interface.

4.1 Uniforming the Type System to the Common Interface and to HPSG Theory

The need to uniform the type system to the common interface has lead to the development of the internal structure of the attributes CONTENT and COTEXT (whose values are of type *cont* and *cotx*, respectively), which belong to the configuration of attributes of the type *loc*. Figure 1 contains the type *loc* in which the values of some attributes have been affected by these changes. Figure 2 shows the changes in detail:

```
loc
     atts category => cat,
          content => cont,
          context => cotx.
```

Fig. 1

```
cont
     atts index => index,
          restr => restr.

cotx
     atts background => &atom _ ,
          examples => &list &atom _ .

index
     atts number => &boolean {sing/plur},
          person => &boolean {'1'/'2'/'3'},
          gender => &boolean{fem/masc},
          usunumb => &boolean {sing/plur}.
```

Fig. 2

The attribute EXAMPLES was introduced in the previous version of the type system as attribute of the types *word* and *lexphrase*, together with attributes such as LEMMA, SENSEID, FORMALITY, etc.. The change of location of this attribute from *word* and *lexphrase* to *cotx* was already shown in Deliverable 3 of the ET-10 Project. The reason is that while the first version of the type system was oriented towards the formalisation of the dictionary entries as such, we then began to work towards producing a more theory-oriented representation.

The types *cont* and *cotx* have been expanded according to HPSG theory (see Pollard and Sag, vol. 1 and 2); this implied the addition of the canonical HPSG attributes INDEX, RESTR and BACKGROUND. However, the insertion of INDEX is not really an addition. In fact, this attribute corresponds to the attribute AGR, which in the old type system was in the configuration of attributes of the type SUBSTANTIVE, as shown in Figure 3:

OLD VERSION

```
substantive:head
     atts predicative => &atom yes/no,
          agr => agr
     subs noun/verb/adj/prep.

agr
     atts number => &boolean {sing/plur},
          person => &boolean {'1'/'2'/'3'},
          gender => &boolean{fem/masc},
          usunumb => &boolean {sing/plur}.
```

Fig. 3

Furthermore, the attribute NFORM has been added in order to conform to HPSG theory (see Figure 4), where the head feature NFORM is used "to distinguish "normal" noun phrases and pronouns [...] from the expletive pronouns *it* and *there* which have very special syntactic properties" (see Pollard and Sag 1987, pag. 62).

```
noun:substantive
     vals maj => n
     atts case => &atom s/o,
          proper => &atom yes/no,
          count => &boolean {co/ma/un}
          part => &atom yes/no,
          nform => &atom norm/it/there
     defs {proper => no part => no}.
```

Fig. 4

Figure 4 shows another intervention which reduces the range of the possible values of the attribute COUNT to 'countable', 'uncountable' and 'mass'. COUNT was previously defined as follows:

```
count => &boolean {co/ma/un/nsing/nplur}
```

but this appeared as a misleading way of encoding the particular kind of grammar code disjunction included in Cobuild (this point has already been discussed in Deliverables 2 and 3). Therefore for now we prefer to assign to COUNT this reduced range of values, with the intention of studying a better way of encoding the particular combinations of grammar codes of Cobuild, which should also respect theoretical requirements.

4.2 Encoding the Information extracted from Cobuild

In the following, we refer to the representation of verb entries. This is because, at the moment, the extraction procedure has been more completely developed on this word class. In fact, the most important integrations and changes are those which refer to the semantic properties of arguments selected by verbs. We therefore introduce an attribute SEMRESTR in the definition of the type *sub_syn_sem*, which is used as value of the attribute SUBCAT, to represent the semantic properties of arguments subcategorized by verbs. In the following we will first discuss the location of this kind of information within the type system, and we will then briefly describe its formalization.

A question which might be addressed at this point relates to the location of this kind of information within the type system: why should the semantic

properties of the arguments selected by a verb be encoded in the SUBCAT list rather than as semantic selection restrictions which instead should be encoded as values of the attributes RESTR in the type *cont*?

The main reasons which lead us to propose this kind of encoding are that, on the one hand, the SUBCAT, which takes a list of (partially specified) signs as its values, also allows us to encode semantic properties of the arguments subcategorized by the verb; on the other hand, the semantic selection restrictions as they are defined in HPSG do not allow us to encode this kind of information. Moreover the relations that are encoded in RESTR are function of the roles played by the arguments with respect to the verb (e.g. 'giver', 'given', 'gift', in Pollard and Sag, vol.2), whereas the properties we are extracting and encoding are inherent to the argument.

We thus propose encoding the information inherent to the arguments of verbs in the SUBCAT list. Our starting point in the old type system is shown in Figure 5, whereas Figure 6 contains the new definitions of the type *sub_syn_sem* together with the types used as values of its attributes. These specifications illustrate the formalization of the information extracted from Cobuild entries.

```
sub_syn_sem:syn_sem
     atts obblig => &atom yes/no.
```

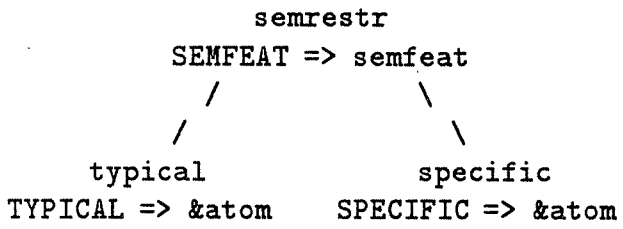
Fig. 5

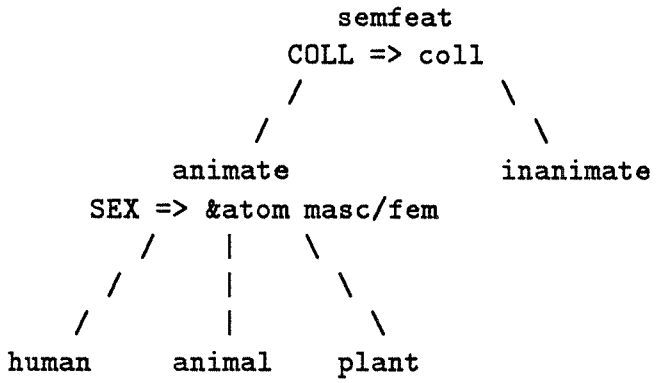
```
sub_syn_sem:syn_sem
     atts obblig => &atom yes/no,
          semrestr => semrestr.

semrestr
     atts semfeat => semfeat
     subs typical/specific.

typical:semrestr
     atts typical => &atom _ .

specific:semrestr
```

```
        atts specific => &atom _ .

semfeat
        atts coll => coll
        subs animate/inanimate.

animate:semfeat
        atts sex => &atom masc/fem
        subs human/animal/plant.
```

Fig. 6

This situation can be also represented as follows:





The attribute SEMRESTR is used to specify the semantic properties inherent to each argument, such as, for example, 'animate', 'human' etc.. At the moment only a part of the information that we are extracting has been considered in the development of the type system. The reason for this is that

whereas up until now we had concentrated our work on TFS on a theoretical study of the system, we have only been able to begin to test our hypotheses on the actual parsed dictionary data very recently, and have thus not yet evaluated in detail all the changes and integrations that it will be necessary to make to our first proposal of representation.

The information contained in Cobuild, which has been classified in the extraction phase by means of the attributes TYPICAL and SPECIFIC (see the section on Attributes and Values of Verbs) is represented in the type system as value of the attributes, identified by the same labels, which are both of type atomic. These attributes are used to define the types *typical* and *specific* respectively, which, as subtypes of *semrestr*, inherit from the common supertype the attribute SEMFEAT. The value of this attribute is used to represent, in a more organized way, the different features derived for the verb arguments, which in the extraction phase are stored in a flat list (see the same section cited above). The features extracted are here represented in part as types and in part as attributes of types. For example, the information stored as "+/-masc" and "+/-fem" is represented by SEX (which can take *masc* or *fem* as values), which is an attribute of the type *animate*; instead the information stored in "+/-human" is represented by expanding the type *animate* with the types ***human***, ***animal*** and ***plant***.

5 Next Steps

In the first three deliverables, the attention of the Pisa group was mainly focussed on the general structure of the Cobuild Student's Dictionary entries with a view to their formalization in TFS. Partly influenced by our previous experience in the derivation of semantic information from other machine readable dictionaries, we concentrated at first on problems which regarded the analysis of the RHS, attempting to define strategies to extract detailed semantic information from this part of the definition. Our first proposal of the possible representation in TFS, at a theoretical level, principally regarded this type of information (see Montemagni et al 1992, Montemagni 1992). However, this study clearly evidenced a number of problems which can be summarised in two main points:

(i) The superordinate and discriminator data found in the Cobuild definitions are considerably different from those of more traditional dictionaries.

In particular, we found that in many cases, depending on the defining formula adopted, it was not possible to identify a superordinate and construct significant taxonomies directly. For example, compare the Cobuild definition for "frenzy", "Someone who is in a frenzy is very excited and violent or

uncontrolled", with that of OALDCE: "violent excitement". In the Cobuild definition, the word has been contextualized and thus the definition itself also gives the typical usage whereas, in OALDCE, this information is given in an example following the definition: "In a frenzy of despair/excitement". From the OALDCE definition, a superordinate (excitement) and a discriminator (violent) can be extracted directly but the contextualization in Cobuild means that the noun "frenzy" has been described in terms of three adjectives "excited", "violent", "uncontrolled" (actually the definition statement regards a person in a frenzied state rather than the word "frenzy" in isolation).

In other cases, when it is possible to derive the superordinate directly from the definition, frequently the term used is so general that it is difficult to extract detailed and useful information from it. For example, compare the way in which "frieze" is defined in these two dictionaries: Cobuild tells us that "A frieze is a long, narrow, decorative feature along the top of a wall". However, "feature" is such a general term, that all the semantic information concerning "frieze" must be derived from the adjectives used to qualify it. Instead, OALDCE describes "frieze" as an "ornamental band or strip along a wall (usu at top), e.g. ...", where the concepts of "long" and "narrow" are carried by the use of band or strip.

These differences in the Cobuild definitions make them interesting but signify that the information that can be extracted from this side of the definition is somewhat different from that extractable from the dictionaries we have analysed previously and imply that different strategies must be studied if meaningful information is indeed to be extracted.

(ii) In any case, even when this kind of semantic information can be easily extracted from the Cobuild definition, it is very difficult to represent it under ALEP as this system, which was not studied specifically for the representation of lexical entries, does not allow inheritance between lexical entries. This problem has been discussed in detail in the reports mentioned above and remains a primary consideration for us.

On the other hand, our first study of the definition data led us to a growing awareness of the importance and potential of the information which was truly specific to the Cobuild dictionary - that information on preferences with regard to syntactic structure and the related semantic and lexical environment which, whenever possible, is given consistently in the LHS of the definition of each lexical item. Encoded in a suitable way, we realized that this kind of syntactic-semantic information could be extremely useful for the lexical component of any NLP system. This led us to the detailed analysis of this part of the definition and the identification of the types of information which could be extracted that has been described here.

At this point, it is our intention to combine in a more general framework

the results of these two somewhat separate lines of investigation. For this reason, we are now studying how links could be created between the semantic preferences identified for the different arguments of the verbs and the corresponding dictionary entries for nouns.

The strategies which we are now considering are essentially of three types. The first will be driven by grammatical information when this is available. For instance, let us consider the Cobuild definition for "pick up" sense 4. The LHS gives us "If you pick up a skill or idea ..." From this, we have extracted "skill" as one of the preferred arguments for "pick up" in this sense and one of the features attached to "skill" is "+count". The dictionary entry for "skill" gives two senses: count and uncount. We can thus automatically select the right sense of "skill" when it is an argument of "pick up". However, using this strategy, it is not always possible to immediately identify the correct sense of an item, as shown by the following example. The definition for "cover" sense 5 begins "If a law covers ..." and our analysis has tagged "law", the required argument for "cover" in this sense, as "+count". In this case, as the dictionary entry gives five senses for law, two of which are indicated as Count Nouns, we are able to restrict our search to these two senses. Some other strategy must be found if the correct one is to be identified automatically.

However, we think that a second strategy based on the semantic features which our analysis has attributed to a given argument is more interesting. For example, with "abdicate" sense 1, "If a king or queen abdicates, ..." we have as specific subjects "king" and "queen" to which the features +hum, +masc, and +hum, +fem, respectively, have been attached (see our semantically analysed entry for "abdicate" in Appendix 3). This will permit us to map directly to the first sense of the dictionary entry for "king" whose superordinate is "man" (in the other senses of "king" the superordinates are (chess) piece, and playing card) and also to the first sense of "queen" where the superordinate is "woman" (the other senses have bee, (chess) piece, and playing card as their superordinates).

A third strategy which we are now considering will involve exploiting the syntactic-semantic features which we have encoded on our verb arguments and encoding the nouns in our lexicon with the same features, trying to infer this information from the RHS. Thus, for example, if for a given verb we know that its subject must be human while its object belongs to the "food" domain, we will then be able to select its potential arguments from those nouns which satisy these conditions in our lexicon. In this way, we hope to be able to derive a very powerful tool from Cobuild definitions for future applications in NLP.

Indeed, connecting the verb arguments or, more generally, the words appearing in co-text1 and co-text2 with the corresponding lexical entries will add a further dimension to our lexical network, that of syntagmatic relations. In this way, the lexical network resulting from our formalization of

the Cobuild dictionary will include two dimensions: that of paradigmatic and that of syntagmatic links. The paradigmatic links such as synonymy, hyponymy, hyperonymy, as well as meronymy, are those usually extracted and formalized from definitions of machine readable dictionaries, and then exploited in the construction of a hierarchical lexicon (see the work already done in Acquilex and other similar projects). By adding the syntagmatic dimension, the lexicon is no longer a hierarchy of isolated words linked by different kinds of paradigmatic relations, but becomes a hierarchy of words inserted in their typical contexts, connected therefore by two different kinds of links. In this way, the context itself has become an access key, with the result of strengthening the possibilities of access to the lexicon, through both the paradigmatic and syntagmatic links. From a theoretical perspective, this solution, which blurrs the traditional decoupling of lexis and syntax, could also be seen as an attempt to formalize the linguistic hypotheses behind the Cobuild dictionary.

6 References

Allport G., Barnbrook G., Sinclair J. (1993a). "The Representation of Cobuild Definitions", Working Paper for ET-10, May 1993.

Allport G., Barnbrook G., Sinclair J. (1993b). "A Grammar of Cobuild Definition Statements", Working Paper for ET-10, June 1993.

ET-10/51: "Deliverable 3. The Cobuild Functional Parser, Pisa TFS Grammar, Bochum BLF Format and Common Interface", Draft Specifications, CEC 1993.

Hanks P. (1987). "Definitions and Explanations", in J.M. Sinclair (ed.), *Looking Up: An Account of the COBUILD Project in Lexical Computing*, Collins COBUILD, Birmingham, pp. 116-136.

Marinai E., Peters C., Picchi E. (1990), "The Pisa Multilingual Lexical Database System", Esprit BRA 3030, Twelve Month Deliverable, ILC- ACQ-2-90, Pisa, 61p.

Montemagni S., Marinai E., Calzolari N., (1992). "Describing Cobuild Lexical Entries as Typed Feature Structures, with Particular Reference to the Semantics of Definitions", Interim Report, CEC Project, ET-10/51 (D2).

Montemagni S. (1992). "Using a Typed Feature structure (TFS) Representation System for Encoding Lexical Knowledge extracted from Dictionary Definitions: Draft Specifications for TFS Descriptions", Deliverable, ET-10/51.

Moon R. (1987). "The Analysis of Meaning", in J.M. Sinclair (ed.), *Looking Up: An Account of the COBUILD Project in Lexical Computing*, Collins COBUILD, Birmingham, pp. 86-103.

Picchi E. (1991). "D.B.T.: A Textual Data Base System", in L. Cignoni and C. Peters (eds.), Computational Lexicology and Lexicography. Special Issue dedicated to Bernard Quemada. I., *Linguistica Computazionale*, Vol VII, 177-205.

Pollard C., Sag I. (1987). *Information-based syntax and semantics* - Vol. 1, Stanford, CSLI (Distributed by University of Chicago Press).

Pollard C., Sag I. (in press). *Head-driven phrase structure grammar*, Stanford, CSLI (Distributed by University of Chicago Press).

Sinclair J. (1987). "Grammar in the Dictionary", in J.M. Sinclair (ed.), *Looking Up: An Account of the COBUILD Project in Lexical Computing*, Collins COBUILD, Birmingham, pp. 104-115.

Sinclair J. (1991). "Words about Words", in J. M. Sinclair (ed.), *Corpus, Concordance, Collocation*, OUP, Oxford, 123-137.

Appendix 1: Technical Details of the Parser

List Manipulation Functions In the following we briefly illustrate the subset of the main functions created for manipulating lists of words. Obviously self-explaining functions are not described since they recall the corresponding Lisp functions. The Lisp-like output format of the Birmingham parser has facilitated this programming style.

- `head( list): listitem`

- `tail( list): list`

- `equal( list, list): boolean`

- `valuecat( list, list): list`
  `concatenation of the two list arguments`

- `empty( list): boolean`

- `wordcount( list): integer`
  `number of words contained in the list argument`

- `nth( integer, list): listitem`
  `n-th element of the list argument`

- `beginwith( list, list): boolean`
  `result is True if the first list argument begins with the second list argument`

- `contain( list, list): boolean`
  `if the first list argument contains the second list argument, the result is the position of the first list argument from which the second list argument starts`

- `after( integer, list): list`
  `the part of the list argument that starts after the position specified by the first argument`

- `upto( integer, list): list`
  `the part of the list argument up to the position specified by the first argument`

- `intersect( list, lists, integer, integer): boolean`
  `if the first argument list intersects one of the lists contained in the second argument set of lists, the result is True. The third integer argument represents the number of the intersecting list in the list set; the fourth integer argument represent the position of the intersecting list with respect to the first argument list`

String Manipulation Functions The subset of simple string manipulation functions is the following:

```
- assignstr( string, string)
     assign the second argument value expression to the first argument string
     variable

- valuecatspc( string, string): string
     concatenation of the first argument string and the second argument string
     separated by a single space

- valuenumcat( string, integer): string
     concatenation of the first argument string and the string obtained from
     the second integer argument
```

Grammar Functions The grammar functions are illustrated below:

```
- PoS( string): string
     part of speech of the complex grammar code represented in the argument
     string

- multicode( string, string, string): boolean
     if the first argument string represents the string of possible grammar
     codes for the entry to be examined, the second argument string will be
     the grammar code actually present in the definition, and the third
     argument string will be the remaining grammar code(s)

- linearize( string, strings, integer)
     the set of resulting strings (the cardinality of which is represented by
     the third integer argument) represented in factorized form in the
     first argument string.


     if string = "the set of particular beliefs, ideas, or prejudices of
               someone"

     then: strings = { "the set of particular beliefs of someone",
                       "the set of particular ideas of someone",
                       "the set of particular prejudices of someone"}
           stringnum = 3
```

Input/Output Structure Manipulation Functions The following auxiliary function subset has been created to treat binary tree structures modelling our i/o structures:

```
- newstruct(): struct
    new node for the tree structure

- putvalueout( string, string)
    the first argument string indicates the output tree path to reach the node
    where the value represented by the second argument string must be placed

- putvaluein( string, string)
    as above for the input tree

- getvalueout( string): string
    the argument string represents the output tree path to reach the node
    containing the desidered value

- getvaluein( string)
    as above for the input tree

- exist( string): boolean
    result is True if the path represented by the argument string
    exists in the input tree

- readinpdef( file): struct
    read data from the specified file to build the input tree structure

- writeoutdef( struct, file)
    write on file the first argument structure

- cleanstruct( struct)
    garbage collection for the memory space used for a tree structure
```

All these functions have been used in the following program chunks which illustrate the main operation of the Parser. We think these functions are now largely self-explanatory.

Main Program

```
main()
{
   int formcount;
   fpr = fopen( "birm.inp", "r");
   fpw = fopen( "sempars.out", "w");
   outputdef = inputdef = NULL;
   for( formcount = 0; !feof( fpr); formcount++)
     {
       inputdef = readinpdef( fpr);
       putvalueout( "def_no", getvalueinp( "def_number"));
       putvalueout( "sense_no", getvalueinp( "sense"));
       putvalueout( "def_type", getvalueinp( "def_type"));
       if(     equal( PoS( getvalueinp( "grammar")), "VERB"))
         elabVERBdef();
       else if( equal( PoS( getvalueinp( "grammar")), "NOUN"))
         elabNOUNdef();
       else if( equal( PoS( getvalueinp( "grammar")), "ADJ"))
         elabADJdef();
       writeoutdef( outputdef, fpw);
       cleanstruct( inputdef);
       cleanstruct( outputdef);
       outputdef = inputdef = NULL;
     }
}
```

Verb Definition Processing The procedure begins by examining the contents of the 'grammar' field in the Birmingham input. The value for this field is used to select the particular template which specifies the preference information which we expect to derive. Whereas, the first block of Attributes shown in the Template, GENERAL_INFO, is common for each verb definition, for each different value of 'grammar' the procedure assigns the appropriate PREFERENCE_INFO template. In the following we show the program segment that treqts the templates. Examples of the different types of PREFERENCE_INFO templates defined so far are given.

```
elabVERBdef(){
   char *grammarcode;
   cotext1 = getvalueinp("lhs-1 co-text1");
   cotext2 = getvalueinp("lhs-1 co-text2");
   grammarcode = getvalueinp("grammar");
   elabentry(grammarcode);
   elabgenus();
   putvalueout("inflection",getvalueinp("lemma"));
   elabgrammarinformation(&grammarcode);
   if(     equal(grammarcode,"VB with OBJ"))
     elabVBwithOBJ();
   else if(equal(grammarcode,"VB"))
     elabVB();
   else if(equal(grammarcode,"VB with OBJ and OBJ"))
     elabVBwithOBJandOBJ();
   else if(equal(grammarcode,"VB with OBJ and TO-INF"))
     elabVBwithOBJandTOINF();
   else if(beginwith(grammarcode,"VB with OBJ and"))
     elabVBwithOBJandPREP();
   else if(equal(grammarcode,"REPORT VB"))
     elabREPORTVB();
   else if(beginwith(grammarcode,"VB with PREP"))
     elabVBwithPREP();
   else if(equal(grammarcode,"PHRASE"))
     elabPHRASE();
   else if(equal(grammarcode,"PHR VB"))
     elabPHRVB();
   else if(equal(grammarcode,"VB with ADJUNCT"))
     elabVBwithADJUNCT();
   else if(beginwith(grammarcode,"VB with 'as'"))
     elabVBwith_as();
   else if(beginwith(grammarcode,"VB with"))
     elabVBwithPREP();
   else if(equal(grammarcode,"ERG VB"))
     elabERGVB();
   else if(equal(grammarcode,"PASSIVE VB"))
     elabPASSIVEVB();
   else if(equal(grammarcode,"PASSIVE VB with ADJUNCT"))
     elabPASSIVEVBwithADJUNCT();
}


elabVBwithOBJ(){
   if(exist("lhs-1 match_to"))
     elabVBtemplate13();
   elabvoice("lhs-1 head1");
   if(equal(getvalueout("voice"),"passive"))
     if(exist("lhs-1 co-text2 match2"))
       elabVBtemplate14();
     else
       elabVBtemplate11();
   else if(exist("lhs-1 co-text2 match3"))
     elabVBtemplate4("preferred:");
   else
     elabVBtemplate1();
}


elabVB()
{
   if(exist("lhs-1 co-text2"))
     elabVBtemplate3("preferred:");
   else
     elabVBtemplate2();
}


elabVBwithOBJandPREP()
{
```

```
   elabVBtemplate4("obligatory:");
}


elabREPORTVB()
{
   if( equal( head( cotext2), "that") ||
      equal( head( cotext2), "who") ||
      equal( head( cotext2), "what"))
    elabVBtemplate5();
   else
    elabVBtemplate6();
}


elabVBwithPREP()
{
   elabVBtemplate3("obligatory:");
}


elabPHRASE()
{
   putvalueout("phrase_type","verbal");
   if(     !exist("lhs-1 co-text2"))
    elabVBtemplate2();
   else if( exist("lhs-1 co-text2 match2"))
    elabVBtemplate6();
   else
    elabVBtemplate3("preferred:");
}


elabPHRVB()
{
   if(     !exist("lhs-1 co-text2"))
    elabVBtemplate2();
   else if( exist("lhs-1 co-text2 match2"))
    elabVBtemplate6();
   else
    elabVBtemplate3("preferred:");
}


elabVBwith_as()
{  elabVBtemplate5();
}


elabERGVB()
{
   if( !empty("lhs-1 co-text2") && !firstmatchflag)
    elabVBtemplate3("preferred:");
   else
    elabVBtemplate6();
}


elabVBwithOBJandOBJ()
{
   elabVBtemplate7();
}


elabVBwithADJUNCT()
{
   elabVBtemplate8();
}


elabPASSIVEVB()
{
   if( exist("lhs-1 co-text2 match2"))
    elabVBtemplate14();
   else
    elabVBtemplate11();
}


elabPASSIVEVBwithADJUNCT()
{
   elabVBtemplate12();
}


elabVBwithOBJandTOINF()
{
   elabVBtemplate10();
```

```
}
```

Verb Definition Templates

```
elabVBtemplate1()
{
  elabvoice("lhs-1 head1");
  elabinference();
  elabsubj_info("lhs-1 co-text1 match1");
  elabobj_info("lhs-1 co-text2 match2");
}


elabVBtemplate2()
{
  elabvoice("lhs-1 head1");
  elabinference();
  elabsubj_info("lhs-1 co-text1 match1");
}


elabVBtemplate3(preptype)
  char *preptype;
{
  elabvoice("lhs-1 head1");
  elabinference();
  elabsubj_info("lhs-1 co-text1 match1");
  if( !exist("lhs-1 co-text2 match2"))
    putvaluein("lhs-1 co-text2 match2",
               after( 1, getvalueinp("lhs-1 co-text2")));
  elabobliq_info("lhs-1 co-text2 match2", preptype);
}


elabVBtemplate4(preptype)
  char *preptype;
{
  elabvoice("lhs-1 head1");
  elabinference();
  elabsubj_info("lhs-1 co-text1 match1");
  elabobj_info("lhs-1 co-text2 match2");
  elabobliq_info("lhs-1 co-text2 match3", preptype);
}


elabVBtemplate5(){
  elabvoice("lhs-1 head1");
  elabinference();
  elabsubj_info("lhs-1 co-text1 match1");
  elabclause_info();
}


elabVBtemplate6()
{
  elabvoice("lhs-1 head1");
  elabinference();
  elabsubj_info("lhs-1 co-text1 match1");
  elabobj_info("lhs-1 co-text2 match2");
}


elabVBtemplate7()
{
  elabvoice("lhs-1 head1");
  elabinference();
  elabsubj_info("lhs-1 co-text1 match1");
  elabobj_info("lhs-1 co-text2 match2");
  elabobj_info("lhs-1 co-text2 match3");
}


elabVBtemplate8()
{
  elabvoice("lhs-1 head1");
  elabinference();
  elabsubj_info("lhs-1 co-text1 match1");
  elabadjunct_info("lhs-1co-text2 match2");
}


elabVBtemplate9()
{
  elabvoice("lhs-1 head1");
```

```
  elabinference();
  elabsubj_info("lhs-1 co-text1 match1");
  elabobj_info("lhs-1 co-text2 match2");
  elabadjunct_info("lhs-1 co-text2 match3");
}


elabVBtemplate10()
{
  elabvoice("lhs-1 head1");
  elabinference();
  elabsubj_info("lhs-1 co-text1 match1");
  elabobj_info("lhs-1 co-text2 match2");
  elabtoinf_info("lhs-1 co-text2");
}


elabVBtemplate11()
{
  elabvoice("lhs-1 head1");
  elabinference();
  elabobj_info("lhs-1 co-text1 match1");
}


elabVBtemplate12()
{
  elabvoice("lhs-1 head1");
  elabinference();
  elabobj_info("lhs-1 co-text1 match1");
  elabadjunct_info("lhs-1 co-text2 match2");
}


elabVBtemplate13()
{
  elabvoice("lhs-1 head1");
  elabobj_info("lhs-1 co-text2 match1");
}


elabVBtemplate14()
{
  elabvoice("lhs-1 head1");
  elabinference();
  elabobj_info("lhs-1 co-text1 match1");
  elabobliq_info("lhs-1 co-text2 match2", "preferred:");
}
```

Noun Definition Processing

```
elabNOUNdef()
{
  char *grammarcode;

  cotext1 = getvalueinp("lhs-1 co-text1 ");
  cotext2  = getvalueinp("lhs-1 co-text2");
  grammarcode = getvalueinp("grammar");
  elabgrammarinformation(&grammarcode);
  elabentry(grammarcode);
  elabgenus();
  elabpurpose();
  if( equal(    grammarcode,"COUNT N"))
   {
    putvalueout("inflection",upto( 2, getvalueinp("lemma")));
    elabform();
   }
  if( equal( grammarcode,"COUNT N with SUPP"))
   elabNOUNwithSUPP();
  else if( beginwith( grammarcode,"COUNT N with"))
   elabNOUNwithPREP();
  else if( equal( getvalueinp("def_type"),"1"))
   elabPHRASENOUN();
  else
   elabNOUN();
}

elabNOUNwithSUPP()
{
  elabNOUNtemplate1();
}


elabNOUN()
{
  elabNOUNtemplate1();
}


elabPHRASENOUN()
{
  elabNOUNtemplate2();
}


elabNOUNwithPREP()
{
  elabNOUNtemplate3();
}
```

Noun Definition Templates

```
elabNOUNtemplate1()
{
  char *modifier;
  int itemnum,
     wordpos;

  if( exist("lhs-1 co-text1"))
   {
    modifier = getvalueinp("lhs-1 co-text1");
    if( intersect8(modifier,&itemnum,&wordpos))
     {
      if( wordcount(modifier) != 1)
        putvalueout("entry_info entry_premods",after( 1, modifier));
     }
    else
     putvalueout("entry_info entry_premods",
              getvalueinp("lhs-1 co-text1"));
   }

  if( exist("lhs-1 co-text2"))
   elabNOUNcolloc("lhs-1 co-text2 match1","preferred:");
}


elabNOUNtemplate2()
{
  putvalueout("phrase_type","verbal");
  elabvoice("lhs-1 co-text1");
  elabinference();
  elabsubj_info("lhs-1 co-text1 match1");
```

```
   if( exist( "lhs-1 co-text2"))
     if( empty( cotext2))
       elabtoinf_info("lhs-1 co-text2 match2");
     else if( exist( "lhs-1 co-text2 match1"))
       elabNOUNobliq("lhs-1 co-text2 match1", "preferred: ");
     else if( exist( "lhs-1 co-text2 match2"))
       elabNOUNobliq("lhs-1 co-text2 match2", "preferred: ");
     else
       elabNOUNobliq("lhs-1 co-text2 match3", "preferred:");
}


elabNOUNtemplate3()
{
   char *modifier;
   int itemnum,
      wordpos;

   if( exist( "lhs-1 co-text1"))
     {
       modifier = getvalueinp("lhs-1 co-text1");
       if( intersect8( modifier, &itemnum, &wordpos))
         {
           if( wordcount( modifier) != 1)
             putvalueout("entry_info entry_premods", after( 1, modifier));
         }
       else
         putvalueout("entry_info entry_premods",
                     getvalueinp("lhs-1 co-text1"));
     }
   elabNOUNcolloc("lhs-1 co-text2 match1", "obligatory: ");
}
```

Adjective Definition Processing

```
elabADJdef()
{
   char *grammarcode;

   cotext1 = getvalueinp( "lhs-1 co-text1 ");
   cotext2 = getvalueinp( "lhs-1 co-text2");
   grammarcode = getvalueinp( "grammar");
   putvalueout( "gram", grammarcode);
   elabentry( grammarcode);
   elabgenus();
   elabgenuscolloc();
   if( wordcount( getvalueinp( "lemma")) > 2)
     putvalueout( "inflection" , upto( 3, getvalueinp( "lemma")));
   if( equal( getvalueinp( "def_type"), "1"))
     elabPHRASEADJ();
   else
     elabADJ();
}


elabADJ()
{
   elabADJtemplate1();
}


elabPHRASEADJ()
{
   elabADJtemplate2();
}
```

Adjective Definition Templates

```
elabADJtemplate1()
{
   if( exist( "lhs-1 co-text2"))
     if( empty( cotext2))
       elabADJcolloc( "lhs-1 co-text2 match1");
     else
       elabADJcolloc( "lhs-1 co-text2");
   else
     elabADJcolloc( "lhs-1 co-text1");
}


elabADJtemplate2()
{
   putvalueout( "phrase_type","verbal");
   elabvoice( "lhs-1 co-text1");
   elabinference();
   elabsubj_info( "lhs-1 co-text1 match1");
   if( exist( "lhs-1 co-text2"))
     if(     exist( "lhs-1 co-text2 match1"))
       elabNOUNobliq( "lhs-1 co-text2 match1", "preferred: ");
     else if( exist( "lhs-1 co-text2 match2"))
       elabNOUNobliq( "lhs-1 co-text2 match2", "preferred: ");
     else
       elabNOUNobliq( "lhs-1 co-text2 match3", "preferred: ");
}
```

```
elabentry( grammarcode)
  char *grammarcode;
{
  putvalueout( "lemma", head( getvalueinp( "lemma")));
  if( equal( PoS( grammarcode), "VERB"))
   if( equal( grammarcode, "PHRASE") ||
       equal( grammarcode, "PHR VB"))
     putvalueout( "entry_info entry",
                valuecatspc( getvalueinp( "lhs-1 head1"),
                             getvalueinp( "lhs-1 head2")));
   else
     putvalueout( "entry_info entry", head( getvalueinp( "lemma")));
  else if( equal( PoS( grammarcode), "NOUN"))
   if( equal( getvalueinp( "def_type"), "1"))
     putvalueout( "entry_info entry",
                valuecatspc( getvalueinp( "lhs-1 co-text1"),
                             getvalueinp( "lhs-1 head1")));
   else
     putvalueout( "entry_info entry", getvalueinp( "lhs-1 head1"));
  else if( equal( PoS( grammarcode), "ADJ"))
   if( equal( getvalueinp( "def_type"), "1"))
     putvalueout( "entry_info entry",
                valuecatspc( getvalueinp( "lhs-1 co-text1"),
                             getvalueinp( "lhs-1 head1")));
   else
     putvalueout( "entry_info entry", getvalueinp( "lhs-1 head1"));
}


elabgenus()
{
  char *genus,
       genia[10][1000],
       genustype[100];
  int genusnum;

  genus = getvalueinp( "rhs-2 superordinate");
  if( empty( genus))
   {
     strcpy( genustype, "genus_info synonym");
     genus = getvalueinp( "rhs-2 synonym");
   }
  else
   strcpy( genustype, "genus_info superordinate");
  linearize( genus, genia, &genusnum);
  while( genusnum > 0)
      {
        putvalueout( valuenumcat( genustype, genusnum), genia[ genusnum-1]);
        genusnum--;
      }
}
```

When the grammar field contains one or more "or" this indicates that there is more than one grammar possibility for this particular sense. In this case, two (or more) templates are selected. and the values extracted from the LHS must then be mapped onto the relevant one. From our examination of the data, it appears that sometimes the LHS context which follows represents the 1st grammar possibility, whereas at other times the second or both possibilities are represented. Once the correct template has been chosen, the procedure fills in the appropriate values on the template.

```
elabgrammarinformation(grammarcode)
  char *(*grammarcode);
{
  char *firstgramm,
       *secondgramm;
  firstgramm = secondgramm = NULL;
  if( multicode( *grammarcode, &firstgramm, &secondgramm))
   {
     putvalueout( "sec_gram", secondgramm);
     *grammarcode = firstgramm;
   }
```

```
  putvalueout( "gram", *grammarcode);
}


elabvoice( infosite)
  char *infosite;
{
  char *head1;
  if( equal( infosite, "lhs-1 co-text1"))
   putvalueout( "voice", "active");
  else
   {
    head1 = getvalueinp( infosite);
    if( contain( head1,"is") ||
        contain( head1,"are"))
     if( contain( head1, nth( 3, getvalueout( "inflection"))))
      putvalueout( "voice", "progressive");
     else
      putvalueout( "voice", "passive");
    else
     putvalueout( "voice", "active");
   }
}


elabinference()
{
  char *deftype,
       *hinge,
       *projection;
  int itemnum,
      wordpos;

  deftype = getvalueout( "def_type");
  if( equal( deftype, "1"))
   {
    hinge = getvalueinp( "op-word hinge");
    match1 = getvalueinp( "lhs-1 co-text1 match1");
    match2 = getvalueinp( "lhs-1 co-text2 match2");
    projection = getvalueinp( "lhs-1 projection");
    if( equal( hinge, "if"))
    if( equal( match1, "you"))
     putvalueout( "inferenence", "possible likely");
    else if( equal( match1, "someone"))
     putvalueout( "inference", "possible negative/unlikely");
    else if( contain( match1, "group") ||
           contain( match1, "people"))
     putvalueout( "inference", "possible collective");
    else if( contain( projection, "you say that") ||
           contain( projection, "you describe"))
     putvalueout( "inference", "subjective");
    else
     putvalueout( "inference", "possible");
    else if( equal( hinge, "when"))
    if( equal( match1, "you"))
     putvalueout( "inference", "inherent likely");
    else if( equal( match1, "someone"))
     putvalueout( "inference", "inherent negative/unlikely");
    else if( contain( match1, "group") ||
           contain( match1, "people"))
     putvalueout( "inference", "inherent collective");
    else
     putvalueout( "inference", "inherent");
  }
}


elabmatch_info( lhsmatch, rhsmatch, match, matchfeature)
  char *lhsmatch,
       *rhsmatch,
       *match,
       *matchfeature;
{
  char subj[10][1000],
       subjqual[10][1000],
       subjmatch[10][1000],
       *countfeature,
       *features,
       *qualifier;
  int itemnum,
      itemlength,
      wordpos,
      itemnumdummy,
      wordposdummy,
      subjnum,
      subjmatchnum;

  linearize( lhsmatch, subj, &subjnum);
```

```
linearize( rhsmatch, subjmatch, &subjmatchnum);
while( subjnum > 0)
    {
     if( intersect2( subj[ subjnum-1], &itemnum, &wordpos) &&
        !equal( nth( wordpos-1, subj[ subjnum-1]), "of"))
       if( intersect4( subj[ subjnum-1], &itemnumdummy, &wordposdummy))
         {
          if( itemnumdummy == 0 || itemnumdummy == 1)
           itemlength = 1;
          else
           itemlength = 2;
          strcpy( subj[ subjnum-1],
                after( wordposdummy+itemlength-1));
          features = list2[itemnum][1];
          assignstr( &countfeature, "");
          if( intersect8( subj[ subjnum-1], &itemnum, &wordpos))
           if( contain( subj[ subjnum-1], "of"))
            strcpy( subj[subjnum-1], after( 1, subj[subjnum-1]));
           else
            {
             strcpy( subj[ subjnum-1], after( 1, subj[ subjnum-1]));
             if( !equal( list8[itemnum][0], "the"))
              assignstr( &countfeature, ", +count");
            }
           else if( equal( getvalueinp( "headi"),
                    nth( 2, getvalueinp( "lemma"))))
            assignstr( &countfeature, ", -count");
           else if( equal( getvalueinp( "headi"),
                    nth( 1, getvalueinp( "lemma"))))
            assignstr( &countfeature, ", +count");
           if( beginwith( subj[subjnum-1], "someone's") ||
               beginwith( subj[subjnum-1], "something's"))
            {
             putvalueout( valuenumcat( valuecat( match, "_premods"),
                                       subjnum),
                          nth( 1, subj[subjnum-1]));
             strcpy( subj[subjnum-1], after( 1, subj[subjnum-1]));
            }
           else if( contain( subj[subjnum-1], "of someone") ||
                    contain( subj[subjnum-1], "of something"))
            {
             putvalueout( valuenumcat( valuecat( match, "_postmods"),
                                       subjnum),
                          after( wordcount( subj[subjnum-1])-2));
             strcpy( subj[subjnum-1],
                     upto( wordcount( subj[subjnum-1])-2,
                           subj[subjnum-1]));
            }
           else if( wordcount( subj[subjnum]) > 1)
            {
             putvalueout( valuenumcat( valuecat( match, "_premods"),
                                       subjnum),
                          upto( wordcount( subj[subjnum])-1,
                                subj[subjnum]));
             strcpy( subj[subjnum], nth( wordcount( subj[subjnum]),
                                         subj[subjnum]));
            }
           putvalueout( valuenumcat( match, subjnum),
                        valuecat( "typical: ", subj[ subjnum-1]));
           if( !contain( features, "+count") &&
               !contain( features, "-count"))
            features = valuecat( features, countfeature);
           putvalueout( valuenumcat( matchfeature, subjnum), features);
      }
     else
      {
       features = list2[itemnum][1];
       putvalueout( valuenumcat( valuecat( match, "_postmods"),
                                 subjnum),
                    after( wordpos, subj[subjnum-1]));
       putvalueout( valuenumcat( matchfeature, subjnum), features);
      }
     else
      {
       if( intersect3( subjmatch[ subjmatchnum-1], &itemnum, &wordpos))
        features = list3[itemnum][1];
       else
        assignstr( &features, "");
       assignstr( &countfeature, "");
       if( intersect8( subj[ subjnum-1], &itemnum, &wordpos))
        {
         if( contain( subj[ subjnum-1], "of"))
          strcpy( subj[subjnum-1], after( 1, subj[subjnum-1]));
         else
          {
           strcpy( subj[ subjnum-1], after( 1, subj[ subjnum-1]));
           if( !equal( list8[itemnum][0], "the"))
            assignstr( &countfeature, ", +count");
```

```
            }
          }
          else if( equal( getvalueinp( "head1"),
                          nth( 2, getvalueinp( "lemma"))))
            assignstr( &countfeature, ", -count");
          else if( equal( getvalueinp( "head1"),
                 nth( 1, getvalueinp( "lemma"))))
            assignstr( &countfeature, ", +count");
          if( beginwith( subj[subjnum-1], "someone's") ||
              beginwith( subj[subjnum-1], "something's"))
           {
            putvalueout( valuenumcat( valuecat( match, "_premods"),
                                      subjnum),
                         nth( 1, subj[subjnum-1]));
            strcpy( subj[subjnum-1], after( 1, subj[subjnum-1]));
          }
          else if( contain( subj[subjnum-1], "of someone") ||
                   contain( subj[subjnum-1], "of something"))
           {
            putvalueout( valuenumcat( valuecat( match, "_postmods"),
                                      subjnum),
                         after( wordcount( subj[subjnum-1])-2,
                                subj[subjnum-1]));
            strcpy( subj[subjnum-1],
                    upto( wordcount( subj[subjnum-1])-2,
                          subj[subjnum-1]));
          }
          else if( wordcount( subj[subjnum-1]) > 1)
           {
            putvalueout( valuenumcat( valuecat( match, "_premods"),
                                      subjnum),
                         upto( wordcount( subj[subjnum-1])-1,
                               subj[subjnum-1]));
            strcpy( subj[subjnum-1], nth( wordcount( subj[subjnum-1]),
                                          subj[subjnum-1]));
          }
          putvalueout( valuenumcat( match, subjnum),
                       valuecat( "specific: ", subj[ subjnum-1]));
          if( !contain( features, "+count") &&
              !contain( features, "-count"))
            features = valuecat( features, countfeature);
          putvalueout( valuenumcat( matchfeature, subjnum), features);
        }
        subjnum--;
        if( subjmatchnum > 1)
          subjmatchnum--;
       }
}


elabsubj_info( infosite)
  char *infosite;
{
  char *lhsmatch,
       *rhsmatch;

  lhsmatch = getvalueinp( infosite);
  if( equal( infosite, "lhs-1 co-text1 match1"))
    rhsmatch = getvalueinp( "rhs-2 match1");
  else
    rhsmatch = getvalueinp( "rhs-2 match2");
  elabmatch_info( lhsmatch, rhsmatch,
              "subj_info subj", "subj_info subj_features");
}


elabobj_info( infosite)
  char *infosite;
{
  char *lhsmatch,
       *rhsmatch;

  lhsmatch = getvalueinp( infosite);
  if( contain( lhsmatch, "another"))
    {
      lhsmatch = getvalueinp( "lhs-1 co-text1 match1");
      rhsmatch = getvalueinp( "rhs-2 match1");
   }
  else if( equal( infosite, "lhs-1 co-text2 match2"))
    rhsmatch = getvalueinp( "rhs-2 match2");
  else if( equal( infosite, "lhs-1 head1"))
    rhsmatch = getvalueinp( "rhs-2 match2");
  else
    rhsmatch = getvalueinp( "rhs-2 match1");
  elabmatch_info( lhsmatch, rhsmatch,
              "obj_info obj", "obj_info obj_features");
}
```

```
elabadjunct_info(infosite)
  char *infosite;
{
  char *lhsmatch,
      *rhsmatch;

  putvalueout("adjunct_info adjunct_prep",
              head(getvalueinp("lhs-1 co-text2")));
  lhsmatch = getvalueinp(infosite);
  if( contain( lhsmatch, "another"))
    {
      lhsmatch = getvalueinp("lhs-1 co-text1 match1");
      rhsmatch = getvalueinp("rhs-2 match1");
    }
  else if( equal( infosite, "lhs-1 co-text2 match2"))
    rhsmatch = getvalueinp("rhs-2 match2");
  else
    rhsmatch = getvalueinp("rhs-2 match3");
  elabmatch_info(lhsmatch, rhsmatch, "adjunct_info adjunct",
                 "adjunct_info adjunct_features");
}


elabtoinf_info(infosite)
  char *infosite;
{
  putvalueout("to-inf_info to-inf", getvalueinp(infosite));
}


elabclause_info(infosite)
  char *infosite;
{
  putvalueout("clause_info clause",
              valuecatspc(valuecatspc(getvalueinp("lhs-1 co-text2"),
                                      getvalueinp("lhs-1 co-text2 match1")),
                          getvalueinp("lhs-1 co-text2 match2")));
}


elabobliq_info(infosite, preptype)
  char *infosite,
       *preptype;
{
  char *lhsmatch,
       *rhsmatch;

  putvalueout("obliq_info obliq_prep",
              valuecat(preptype, head(getvalueinp("lhs-1 co-text2"))));
  lhsmatch = getvalueinp(infosite);
  if( contain( lhsmatch, "another"))
    {
      lhsmatch = getvalueinp("lhs-1 co-text1 match1");
      rhsmatch = getvalueinp("rhs-2 match1");
    }
  else if( equal( infosite, "lhs-1 co-text2 match2"))
    rhsmatch = getvalueinp("rhs-2 match2");
  else
    rhsmatch = getvalueinp("rhs-2 match3");
  elabmatch_info(lhsmatch, rhsmatch, "obliq_info obliq",
                 "obliq_info obliq_features");
}


elabNOUNobliq(infosite, preptype)
  char *infosite,
       *preptype;
{  char *lhsmatch,
       *rhsmatch;

  putvalueout("obliq_info obliq_prep",
              valuecat(preptype, head(getvalueinp("lhs-1 co-text2"))));
  lhsmatch = getvalueinp(infosite);
  if(     equal( infosite, "lhs-1 co-text2 match1"))
    rhsmatch = getvalueinp("rhs-2 match1");
  else if( equal( infosite, "lhs-1 co-text2 match2"))
    rhsmatch = getvalueinp("rhs-2 match2");
  else
    rhsmatch = getvalueinp("rhs-2 match3");
  elabmatch_info(lhsmatch, rhsmatch, "obliq_info obliq",
                 "obliq_info obliq_features");
}


elabNOUNcolloc(infosite, preptype)
  char *infosite,
       *preptype;
```

```
{
  char *lhsmatch,
      *rhsmatch;

  putvalueout("colloc_info colloc_prep",
            valuecat(preptype, head(getvalueinp("lhs-1 co-text2"))));
  lhsmatch = getvalueinp(infosite);
  if(     equal(infosite,"lhs-1 co-text2 match1"))
    rhsmatch = getvalueinp("rhs-2 match1");
  else if(equal(infosite,"lhs-1 co-text2 match2"))
    rhsmatch = getvalueinp("rhs-2 match2");
  else
    rhsmatch = getvalueinp("rhs-2 match3");
  elabmatch_info(lhsmatch, rhsmatch, "colloc_info colloc",
              "colloc_info colloc_features");
}


elabpurpose()
{
  int wordpos1,
     wordpos2,
     purposenum;
  char *discriminator,
      *discrchunk,
      purpose[10][1000];

  discriminator = getvalueinp("rhs-2 discriminator");
  if( ( wordpos1 = contain(discriminator,"used , for")) ||
     ( wordpos1 = contain(discriminator,"used , to")))
    {
     if( ( wordpos2 = contain(after(wordpos1+2,discriminator),",")))
       discrchunk = upto(wordpos2-1,after(wordpos1+2,discriminator));
     else
       discrchunk = upto(wordcount(discriminator),
                         after(wordpos1+2,discriminator));
     linearize(discrchunk,purpose,&purposenum);
     while(purposenum > 0)
         {
          putvalueout(valuenumcat("genus_info purpose",purposenum),
                      purpose[purposenum-1]);
          purposenum--;
         }
    }
}


elabform()
{
  char *str1,
      *str2;

  str1 = getvalueinp("lhs-1 head1");
  str2 = nth(2,getvalueinp("lemma"));
  if(equal(++str1,++str2))
    putvalueout("form","plural");
}


elabgenuscolloc()
{
  int wordpos1,
     wordpos2,
     purposenum;
  char *discriminator,
      *discrchunk,
      purpose[10][1000];

  discriminator = getvalueinp("rhs-2 discriminator");
  if(wordpos1 = contain(discriminator,"in"))
    {
     discrchunk = upto(wordcount(discriminator),
                       after(wordpos1,discriminator));
     linearize(discrchunk,purpose,&purposenum);
     while(purposenum > 0)
         {
          putvalueout(valuenumcat("genus_info colloc",purposenum),
                      purpose[purposenum-1]);
          purposenum--;
         }
    }
}


elabADJcolloc(infosite)
  char *infosite;
{
  char *valuelist,
```

```
    obj[10][1000],
    *features;
int objnum,
   itemnum,
   wordpos;

valuelist = getvalueinp( infosite);
if( equal( infosite, "lhs-1 co-text1") &&
   exist( "lhs-1 co-text1"))
 {
   linearize( valuelist, obj, &objnum);
   while( objnum > 0)
       {
        if( intersect2( obj[ objnum-1], &itemnum, &wordpos))
         {
           features = list2[itemnum][1];
           putvalueout( valuenumcat( "colloc_info colloc_postmods",
                                     objnum),
                        after( wordpos, obj[objnum-1]));
           putvalueout( valuenumcat( "colloc_info colloc_features",
                                     objnum),
                        features);
         }
        objnum--;
       }
 }
else
 {
   linearize( valuelist, obj, &objnum);
   while( objnum > 0)
       {
        if( intersect2( obj[ objnum-1], &itemnum, &wordpos))
         {
           features = list2[itemnum][1];
           putvalueout( valuenumcat( "colloc_info colloc_postmods",
                                     objnum),
                        after( wordpos, obj[objnum-1]));
           putvalueout( valuenumcat( "colloc_info colloc_features",
                                     objnum),
                        features);
         }
        else
          putvalueout( valuenumcat( "colloc_info colloc", objnum),
                       valuecat( "specific: ", obj[ objnum-1]));
        objnum--;
       }
 }
}
```

Appendix 2: Main Search Lists

```
char *list2[19][2] = { { "living thing",    "+anim, +count"},
                       { "you",             "+anim, +hum"},
                       { "someone",         "+anim, +hum"},
                       { "man",             "+anim, +hum, +masc, +count"},
                       { "woman",           "+anim, +hum, +fem, +count"},
                       { "person",          "+anim, +hum, +count"},
                       { "somebody",        "+anim, +hum"},
                       { "group of people", "+anim, +hum, +coll"},
                       { "number of people","+anim, +hum, +coll"},
                       { "group",           "+anim, +hum, +coll"},
                       { "people",          "+anim, +hum, +coll"},
                       { "animal",          "+anim, -hum, +animal, +count"},
                       { "insect",          "+anim, -hum, +animal, +count"},
                       { "plant",           "+anim, -hum, +plant, +count"},
                       { "thing",           "-anim"},
                       { "place",           "+loc"},
                       { "things",          "-anim, +pl"}};

char *list3[3][2]   = { { "he",  "+anim, +hum, +masc"},
                        { "she", "+anim, +hum, +fem"},
                        { "it",  "-anim"}};

char *list4[4][2]   = { { "esp",        ""},
                        { "especially", ""},
                        { "for example",""},
                        { "such as",    ""}};

char *list7[3][2]   = { { ", or", ""},
                        { ",",    ""},
                        { "or",   ""}};

char *list8[4][2]   = { { "a",   ""},
                        { "an",  ""},
                        { "one", ""},
                        { "the", ""}};
```

Appendix 3: First Results of the Parser

```
def_no          : 5
sense_no        : 1
def_type        : 3
gram            : COUNT N
lemma           : abacus
entry           : abacus
genus_info      : prov_superordinate1 : frame
                  purpose1            : counting
inflection      : abacus abacuses

def_no          : 13
sense_no        : 1
def_type        : 3
gram            : COUNT N
lemma           : abbot
entry           : abbot
genus_info      : prov_superordinate1 : monk
inflection      : abbot abbots

def_no          : 14
sense_no        : 1
def_type        : 1
lemma           : abbreviate
entry           : abbreviate
genus_info      : prov_superordinate1 : make
inflection      : abbreviate abbreviates abbreviating abbreviated
gram            : VB with OBJ
voice           : active
inference       : possible likely
subj_info       : subj_features1    : +anim, +hum
obj_info        : obj2              : specific: speech
                  obj_features2     : -anim
                  obj_premods1      : piece of
                  obj1              : specific: writing
                  obj_features1     : -anim

def_no          : 17
sense_no        : 1
def_type        : 1
lemma           : abdicate
entry           : abdicate
genus_info      : prov_synonym1     : resigns
inflection      : abdicate abdicates abdicating abdicated
gram            : VB
voice           : active
inference       : possible
subj_info       : subj2             : specific: queen
                  subj_features2    : +anim, +hum, +fem, +count
                  subj1             : specific: king
                  subj_features1    : +anim, +hum, +masc, +count

def_no          : 22
sense_no        : 1
def_type        : 1
lemma           : abet
entry           : abet
genus_info      : prov_superordinate2 : encourage
                  prov_superordinate1 : help
inflection      : abet abets abetting abetted
gram            : VB with OBJ
voice           : active
inference       : possible likely
subj_info       : subj_features1    : +anim, +hum
obj_info        : obj_features1     : +anim, +hum

def_no          : 23
sense_no        : 1
def_type        : 1
lemma           : abhor
entry           : abhor
genus_info      : prov_superordinate1 : hate
inflection      : abhor abhors abhorring abhorred
gram            : VB with OBJ
voice           : active
inference       : possible likely
subj_info       : subj_features1    : +anim, +hum
obj_info        : obj_features1     : -anim

def_no          : 25
sense_no        : 1
def_type        : 1
lemma           : abide
entry           : can't abide
genus_info      : prov_superordinate1 : dislike
```

```
inflection        : abide abides abiding abided
gram             : PHRASE
phrase_type       : verbal
voice            : active
inference         : possible likely
subj_info         : subj_features1     : +anim, +hum
obj_info          : obj_features1      : -anim

def_no           : 26
sense_no          : 2
def_type          : 1
lemma            : abide
entry            : abide
genus_info        : prov_superordinate1 : continues
inflection        : abide abides abiding abided
gram             : VB
voice            : active
inference         : possible
subj_info         : subj_features1     : -anim

def_no           : 27
sense_no          : 2
def_type          : 1
lemma            : abide
entry            : abide by
genus_info        : prov_superordinate1 : do
inflection        : abide abides abiding abided
gram             : PHR VB
voice            : active
inference         : possible likely
subj_info         : subj_features1     : +anim, +hum
obj_info          : obj3               : specific: decision
                  obj_features3       : -anim, +count
                  obj2                : specific: agreement
                  obj_features2       : -anim, +count
                  obj1                : specific: law
                  obj_features1       : -anim, +count

def_no           : 73
sense_no          : 2
def_type          : 3
gram             : SING N
lemma            : absence
entry            : absence
genus_info        : prov_superordinate1 : fact
postcolloc_info    : colloc_prep        : preferred: of
                  colloc_features1    : -anim

def_no           : 88
sense_no          : 4
def_type          : 1
lemma            : absorb
entry            : absorb
genus_info        : prov_synonym1      : learn and understand
inflection        : absorb absorbs absorbing absorbed
gram             : VB with OBJ
voice            : active
inference         : possible likely
subj_info         : subj_features1     : +anim, +hum
obj_info          : obj1               : specific: information
                  obj_features1       : -anim

def_no           : 92
sense_no          : 2
def_type          : 3
gram             : UNCOUNT N with SUPP
lemma            : absorption
entry            : Absorption
genus_info        : prov_synonym1      : the action of absorbing something

def_no           : 95
sense_no          : 1
def_type          : 3
sec_gram          : UNCOUNT N
gram             : COUNT N
lemma            : abstention
entry            : abstention
genus_info        : prov_superordinate1 : act of not voting
inflection        : abstention abstentions

def_no           : 96
sense_no          : 1
def_type          : 3
gram             : UNCOUNT N
lemma            : abstinence
entry            : Abstinence
genus_info        : prov_superordinate1 : the practice of not having something
```

```
def_no           : 100
sense_no         : 4
def_type         : 3
gram             : COUNT N
lemma            : abstract
entry            : abstract
genus_info       : prov_superordinate1 : piece of writing
inflection       : abstract abstracts
postcolloc_info  : colloc_prep       : preferred: of
                   colloc2           : specific: speech
                   colloc_features2  : -anim, +count
                   colloc1           : specific: article
                   colloc_features1  : -anim, +count

def_no           : 105
sense_no         : 1
def_type         : 3
gram             : SING N
lemma            : abundance
entry            : abundance
genus_info       : prov_superordinate1 : quantity of
postcolloc_info  : colloc_prep       : preferred: of
                   colloc_features1  : -anim

def_no           : 109
sense_no         : 1
def_type         : 3
gram             : UNCOUNT N
lemma            : abuse
entry            : Abuse
genus_info       : prov_superordinate1 : things

def_no           : 111
sense_no         : 3
def_type         : 3
gram             : UNCOUNT N with SUPP
lemma            : abuse
entry            : Abuse
genus_info       : prov_superordinate1 : treatment of
postcolloc_info  : colloc_prep       : preferred: of
                   colloc_features1  : +anim, +hum

def_no           : 112
sense_no         : 4
def_type         : 1
lemma            : abuse
entry            : abuse
genus_info       : prov_superordinate1 : use
inflection       : abuse abuses abusing abused
gram             : VB with OBJ
voice            : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj_features1      : -anim

def_no           : 113
sense_no         : 5
def_type         : 3
gram             : UNCOUNT N
lemma            : abuse
entry            : Abuse
genus_info       : prov_superordinate1 : the use of
postcolloc_info  : colloc_prep       : preferred: of
                   colloc_features1  : -anim

def_no           : 115
sense_no         : 1
def_type         : 3
gram             : ADJ
lemma            : abysmal
entry            : Abysmal
genus_info       : prov_superordinate2 : poor
                   prov_superordinate1 : bad
                   colloc1           : quality

def_no           : 125
sense_no         : 1
def_type         : 1
lemma            : accelerate
entry            : accelerate
genus_info       : prov_synonym1      : increases
inflection       : accelerate accelerates accelerating accelerated
gram             : VB
voice            : active
inference        : inherent
subj_info        : subj_postmods2     : of something
                   subj2              : specific: speed
                   subj_features2     : -anim
```

```
                subj_postmods1     : of something
                subj1            : specific: rate
                subj_features1     : -anim

def_no          : 126
sense_no        : 1
def_type        : 3
gram            : COUNT N
lemma           : accelerator
entry           : accelerator
genus_info       : prov_superordinate1 : pedal
inflection       : accelerator accelerators
postcolloc_info   : colloc_prep       : preferred: in
                colloc1          : specific: vehicle
                colloc_features1   : +count

def_no          : 130
sense_no        : 4
def_type        : 1
lemma           : accent
entry           : accent
genus_info       : prov_synonym1      : emphasize
inflection       : accent  accents accenting accented
gram            : VB with OBJ
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj_premods2       : musical
                obj2             : specific: note
                obj_features2      : -anim, +count
                obj1             : specific: word
                obj_features1      : -anim, +count

def_no          : 132
sense_no        : 1
def_type        : 1
lemma           : accept
entry           : accept
genus_info       : prov_superordinate1 : agree
inflection       : accept accepts accepting accepted
sec_gram         : VB
gram            : VB with OBJ
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj_postmods1      : that you have been offered
                obj_features1      : -anim

def_no          : 133
sense_no        : 2
def_type        : 1
lemma           : accept
entry           : accept
genus_info       : prov_superordinate1 : agree
inflection       : accept accepts accepting accepted
gram            : VB with OBJ
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj_premods2       : someone's
                obj2             : specific: suggestion
                obj_premods1       : someone's
                obj1             : specific: advice

def_no          : 134
sense_no        : 3
def_type        : 1
lemma           : accept
entry           : accept
genus_info       : prov_synonym1      : believe
inflection       : accept accepts accepting accepted
gram            : REPORT VB
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj2             : specific: statement
                obj_features2      : -anim, +count
                obj1             : specific: story
                obj_features1      : -anim, +count

def_no          : 143
sense_no        : 2
def_type        : 3
gram            : UNCOUNT N
lemma           : access
entry           : Access
genus_info       : prov_superordinate2: the right
                prov_superordinate1 : the opportunity
```

```
def_no          : 154
sense_no        : 1
def_type        : 1
lemma           : acclaim
entry           : acclaim
genus_info       : prov_superordinate1 : are praised
inflection       : acclaim acclaims acclaiming acclaimed
gram            : VB with OBJ
voice           : passive
inference        : possible
obj_info         : obj_features2      : -anim
                  obj_features1      : +anim, +hum

def_no          : 168
sense_no        : 1
def_type        : 1
lemma           : accomplish
entry           : accomplish
genus_info       : prov_superordinate1 : succeed
inflection       : accomplish accomplishes accomplishing accomplished
gram            : VB with OBJ
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj_features1      : -anim

def_no          : 171
sense_no        : 2
def_type        : 3
gram            : UNCOUNT N with SUPP
lemma           : accomplishment
entry           : accomplishment
genus_info       : prov_synonym1      : fact of finishing
postcolloc_info   : colloc_prep       : preferred: of
                  colloc2           : specific: plan
                  colloc_features2   : -anim, +count
                  colloc1           : specific: task
                  colloc_features1   : -anim, +count

def_no          : 190
sense_no        : 1
def_type        : 1
lemma           : account
entry           : account for
genus_info       : prov_superordinate1 : explain
inflection       : account accounts accounting accounted
gram            : PHR VB
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj_features1      : -anim

def_no          : 212
sense_no        : 1
def_type        : 1
lemma           : achieve
entry           : achieve
genus_info       : prov_superordinate1 : succeed
inflection       : achieve achieves achieving achieved
gram            : VB with OBJ
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj_premods2       : particular
                  obj2              : specific: effect
                  obj_features2      : -anim, +count
                  obj_premods1       : particular
                  obj1              : specific: aim
                  obj_features1      : -anim, +count

def_no          : 221
sense_no        : 1
def_type        : 1
lemma           : acknowledge
entry           : acknowledge
genus_info       : prov_superordinate1 : is accepted
inflection       : acknowledge acknowledges acknowledging acknowledged
gram            : VB with OBJ
voice           : passive
inference        : possible
obj_info         : obj2              : specific: situation
                  obj_features2      : -anim, +count
                  obj1              : specific: fact
                  obj_features1      : -anim, +count

def_no          : 233
sense_no        : 1
```

```
def_type        : 1
lemma           : acquaint
entry           : acquaint
genus_info       : prov_superordinate1 : tell
inflection       : acquaint acquaints acquainting acquainted
gram            : VB with OBJ and 'with'
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj_features1      : +anim, +hum
obliq_info       : obliq_prep         : obligatory: with
                  obliq_features1    : -anim

def_no          : 239
sense_no         : 1
def_type         : 1
lemma           : acquiesce
entry           : acquiesce
genus_info       : prov_superordinate1 : agree
inflection       : acquiesce acquiesces acquiescing acquiesced
gram            : VB
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obliq_info       : obliq_prep         : preferred: to
                  obliq_features1    : -anim

def_no          : 262
sense_no         : 1
def_type         : 1
lemma           : act
entry           : act
genus_info       : prov_superordinate1 : do something
inflection       : act acts acting acted
gram            : VB
voice           : active
inference        : inherent likely
subj_info        : subj_features1     : +anim, +hum

def_no          : 265
sense_no         : 4
def_type         : 1
lemma           : act
entry           : act
genus_info       : prov_superordinate1 : have a part
inflection       : act acts acting acted
sec_gram         : VB with OBJ
gram            : VB
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obliq_info       : obliq_prep         : preferred: in
                  obliq2             : specific: film
                  obliq_features2    : -anim, +count
                  obliq1             : specific: play
                  obliq_features1    : -anim, +count
def_no          : 272
sense_no         : 9
def_type         : 1
lemma           : act
entry           : is acting up
genus_info       : prov_superordinate2 : are not behaving
                  prov_superordinate1 : are not working
inflection       : act acts acting acted
gram            : PHR VB
voice           : progressive
inference        : possible
subj_info        : subj_features2     : -anim
                  subj_features1     : +anim, +hum

def_no          : 276
sense_no         : 2
def_type         : 3
gram            : COUNT N
lemma           : action
entry           : action
genus_info       : prov_superordinate1 : something
inflection       : action actions

def_no          : 337
sense_no         : 4
def_type         : 1
lemma           : address
entry           : address
genus_info       : prov_superordinate1 : give a speech
inflection       : address addresses addressing addressed
gram            : VB with OBJ
```

```
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj_postmods1      : of people
                  obj_features1      : +anim, +hum, +coll

def_no          : 344
sense_no         : 1
def_type         : 1
lemma           : adhere
entry           : adhere
genus_info       : prov_superordinate1 : sticks
inflection       : adhere adheres adhering adhered
gram            : VB with PREP 'to'
voice           : active
inference        : possible
subj_info        : subj1              : specific: substance
                  subj_features1     : -anim, +count
obliq_info       : obliq_prep          : obligatory: to
                  obliq2             : specific: object
                  obliq_features2    : -anim, +count
                  obliq1             : specific: surface
                  obliq_features1    : -anim, +count

def_no          : 346
sense_no         : 3
def_type         : 1
lemma           : adhere
entry           : adhere
genus_info       : prov_synonym1      : support
inflection       : adhere adheres adhering adhered
gram            : VB with PREP 'to'
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obliq_info       : obliq_prep         : obligatory: to
                  obliq2             : specific: belief
                  obliq_features2    : -anim, +count
                  obliq1             : specific: opinion
                  obliq_features1    : -anim, +count

def_no          : 354
sense_no         : 1
def_type         : 1
lemma           : adjoin
entry           : adjoin
genus_info       : prov_superordinate1 : are
inflection       : adjoin adjoins adjoining adjoined
gram            : VB with OBJ
voice           : active
inference        : possible
subj_info        : subj3              : specific: object
                  subj_features3     : +count
                  subj_features2     : +loc
                  subj1              : specific: room
                  subj_features1     : +count
obj_info         : obj3               : specific: object
                  obj_features3      : +count
                  obj_features2      : +loc
                  obj1               : specific: room
                  obj_features1      : +count

def_no          : 355
sense_no         : 1
def_type         : 1
lemma           : adjourn
entry           : adjourn
genus_info       : prov_superordinate1 : is stopped
inflection       : adjourn adjourns adjourning adjourned
gram            : ERG VB
voice           : active
inference        : possible
subj_info        : subj2              : specific: trial
                  subj_features2     : -anim, +count
                  subj1              : specific: meeting
                  subj_features1     : -anim, +count

def_no          : 356
sense_no         : 1
def_type         : 1
lemma           : adjudicate
entry           : adjudicate
genus_info       : prov_superordinate1 : make an official decision
inflection       : adjudicate adjudicates adjudicating adjudicated
sec_gram         : VB with OBJ
gram            : VB
voice           : active
inference        : possible likely
```

```
subj_info          : subj_features1     : +anim, +hum
obliq_info         : obliq_prep         : preferred: on
                   obliq2             : specific: problem
                   obliq_features2    : -anim, +count
                   obliq1             : specific: dispute
                   obliq_features1    : -anim, +count

def_no             : 380
sense_no           : 1
def_type           : 1
lemma              : admit
entry              : admit
genus_info         : prov_superordinate1: agree
inflection         : admit admits admitting admitted
sec_gram           : VB with -ING or PREP 'to'
gram               : REPORT VB
voice              : active
inference          : possible likely
subj_info          : subj_features1     : +anim, +hum
obj_info           : obj_postmods3      : embarrassing
                   obj_features3      : -anim
                   obj_postmods2      : unpleasant
                   obj_features2      : -anim
                   obj_postmods1      : bad
                   obj_features1      : -anim

def_no             : 390
sense_no           : 2
def_type           : 1
lemma              : adopt
entry              : adopt
genus_info         : prov_superordinate1: begin
inflection         : adopt adopts adopting adopted
gram               : VB with OBJ
voice              : active
inference          : possible likely
subj_info          : subj_features1     : +anim, +hum
obj_info           : obj_premods3       : way of
                   obj3               : specific: behaving
                   obj_features3      : -anim
                   obj2               : specific: position
                   obj_features2      : -anim, +count
                   obj1               : specific: attitude
                   obj_features1      : -anim, +count

def_no             : 393
sense_no           : 1
def_type           : 1
lemma              : adore
entry              : adore
genus_info         : prov_synonym1      : love and admire
inflection         : adore adores adoring adored
gram               : VB with OBJ
voice              : active
inference          : possible likely
subj_info          : subj_features1     : +anim, +hum
obj_info           : obj_features1      : +anim, +hum

def_no             : 394
sense_no           : 2
def_type           : 1
lemma              : adore
entry              : adore
genus_info         : prov_superordinate1: like
inflection         : adore adores adoring adored
gram               : VB with OBJ
voice              : active
inference          : possible likely
subj_info          : subj_features1     : +anim, +hum
obj_info           : obj_features1      : -anim

def_no             : 396
sense_no           : 2
def_type           : 1
lemma              : adorn
entry              : adorn
genus_info         : prov_superordinate1: make
inflection         : adorn adorns adorning adorned
gram               : VB with OBJ
voice              : active
inference          : possible
subj_info          : subj_features1     : -anim, +pl
obj_info           : obj_features1      : +loc

def_no             : 407
sense_no           : 2
def_type           : 1
lemma              : advance
```

```
entry           : advance
genus_info       : prov_superordinate1 : make progress
inflection       : advance advances advancing advanced
gram            : VB
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obliq_info       : obliq_prep         : preferred: in
                  obliq_postmods1    : you are doing
                  obliq_features1    : -anim

def_no          : 514
sense_no         : 3
def_type         : 1
lemma           : age
entry           : age
genus_info       : prov_superordinate2 : seem
                  prov_superordinate1 : become
inflection       : age ages ageing UISeor UIVeaging UIFeaged
gram            : ERG VB
voice           : active
inference        : inherent negative/unlikely
subj_info        : subj_features1     : +anim, +hum

def_no          : 529
sense_no         : 1
def_type         : 1
lemma           : aggravate
entry           : aggravate
genus_info       : prov_superordinate1 : make
inflection       : aggravate aggravates aggravating aggravated
gram            : VB with OBJ
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj1               : specific: situation
                  obj_features1      : -anim, +count

def_no          : 540
sense_no         : 1
def_type         : 1
lemma           : agitate
entry           : agitate
genus_info       : prov_superordinate1 : talk and campaign
inflection       : agitate agitates agitating agitated
gram            : VB
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obliq_info       : obliq_prep         : preferred: for
                  obliq_features1    : -anim

def_no          : 553
sense_no         : 4
def_type         : 1
lemma           : agree
entry           : agree
genus_info       : prov_synonym1      : approve of
inflection       : agree agrees agreeing agreed
gram            : VB with 'with'
voice           : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obliq_info       : obliq_prep         : obligatory: with
                  obliq2             : specific: suggestion
                  obliq_features2    : -anim, +count
                  obliq1             : specific: action
                  obliq_features1    : -anim, +count

def_no          : 558
sense_no         : 1
def_type         : 3
gram            : COUNT N
lemma           : agreement
entry           : agreement
genus_info       : prov_superordinate1 : decision
inflection       : agreement agreements

def_no          : 570
sense_no         : 3
def_type         : 1
lemma           : aid
entry           : aid
genus_info       : prov_synonym1      : help
inflection       : aid aids aiding aided
gram            : VB with OBJ
voice           : active
inference        : possible likely
```

```
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj2               : specific: organization
                 obj_features2        : +count
                 obj_features1        : +anim, +hum, +count

def_no           : 591
sense_no         : 5
def_type         : 1
lemma            : air
entry            : air
genus_info       : prov_superordinate1 : make
inflection       : air airs airing aired
gram             : VB with OBJ
voice            : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj_premods1       : your
                 obj1                 : specific: opinions

def_no           : 592
sense_no         : 6
def_type         : 1
lemma            : air
entry            : air
genus_info       : prov_superordinate1 : let fresh air
inflection       : air airs airing aired
gram             : VB with OBJ
voice            : active
inference        : possible likely
subj_info        : subj_features1     : +anim, +hum
obj_info         : obj1               : specific: room
                 obj_features1        : -anim, +count

def_no           : 625
sense_no         : 3
def_type         : 1
lemma            : alarm
entry            : alarm
genus_info       : prov_superordinate1 : makes
inflection       : alarm alarms alarming alarmed
gram             : VB with OBJ
voice            : active
inference        : possible
subj_info        : subj_features1     : -anim
obj_info         : obj_features1      : +anim, +hum

def_no           : 662
sense_no         : 4
def_type         : 1
lemma            : alight
entry            : alight
genus_info       : prov_superordinate1 : get
inflection       : alight alights alighting alighted
gram             : VB
voice            : active
inference        : inherent likely
subj_info        : subj_features1     : +anim, +hum
obliq_info       : obliq_prep         : preferred: from
                 obliq2               : specific: bus
                 obliq_features2      : -anim, +count
                 obliq1               : specific: train
                 obliq_features1      : -anim, +count

def_no           : 663
sense_no         : 1
def_type         : 1
lemma            : align
entry            : align
genus_info       : prov_superordinate1 : support
inflection       : align aligns aligning aligned
sec_gram         : REFL VB
gram             : PASSIVE VB
voice            : passive
inference        : possible likely
obj_info         : obj_features1      : +anim, +hum
obliq_info       : obliq_prep         : preferred: with
                 obliq_features1      : +anim, +hum, +coll

def_no           : 699
sense_no         : 1
def_type         : 1
lemma            : alleviate
entry            : alleviate
genus_info       : prov_superordinate1 : makes
inflection       : alleviate alleviates alleviating alleviated
gram             : VB with OBJ
voice            : active
inference        : possible
```

```
subj_info        : subj_features1     : -anim
obj_info         : obj2               : specific: suffering
                  obj_features2       : -anim
                  obj1               : specific: pain
                  obj_features1       : -anim

def_no           : 920
sense_no         : 1
def_type         : 3
gram             : COUNT N
lemma            : animal
entry            : animal
genus_info       : prov_superordinate1: creature
inflection       : animal animals

def_no           : 1188
sense_no         : 1
def_type         : 3
gram             : COUNT N
lemma            : area
entry            : area
genus_info       : prov_superordinate3: the world
                  prov_superordinate2: a country
                  prov_superordinate1: part of a city
inflection       : area areas

def_no           : 1770
sense_no         : 1
def_type         : 3
gram             : COUNT N
lemma            : bag
entry            : bag
genus_info       : prov_superordinate1: container
                  purpose1           : carry things
inflection       : bag bags

def_no           : 2391
sense_no         : 1
def_type         : 3
gram             : COUNT N
lemma            : bird
entry            : bird
genus_info       : prov_superordinate1: creature
inflection       : bird birds

def_no           : 2655
sense_no         : 1
def_type         : 3
gram             : COUNT N
lemma            : board
entry            : board
genus_info       : prov_superordinate1: piece of wood
                  purpose1           : a particular purpose
inflection       : board boards

def_no           : 2670
sense_no         : 1
def_type         : 3
gram             : COUNT N
lemma            : boat
entry            : boat
genus_info       : prov_superordinate1: vehicle
inflection       : boat boats

def_no           : 3352
sense_no         : 1
def_type         : 3
gram             : COUNT N
lemma            : building
entry            : building
genus_info       : prov_superordinate1: structure
inflection       : building buildings

def_no           : 5580
sense_no         : 1
def_type         : 3
gram             : COUNT N
lemma            : container
entry            : container
genus_info       : prov_superordinate1: something
                  purpose2           : store things
                  purpose1           : hold
inflection       : container containers

def_no           : 6001
sense_no         : 5
def_type         : 1
lemma            : cover
```

```
entry              : cover
genus_info          : prov_superordinate1 : applies
inflection          : cover covers covering covered
gram               : VB with OBJ
voice              : active
inference           : possible
subj_info           : subj1              : specific: law
                    subj_features1       : -anim, +count
obj_info            : obj3               : specific: situations
                    obj_features2        : -anim, +pl
                    obj_premods1         : particular set of
                    obj1                 : specific: people

def_no             : 6002
sense_no            : 6
def_type            : 1
lemma              : cover
entry              : cover
genus_info          : prov_superordinate1 : discuss
inflection          : cover covers covering covered
gram               : VB with OBJ
voice              : active
inference           : possible likely
subj_info           : subj_features1     : +anim, +hum
obj_info            : obj_premods1       : particular
                    obj1                 : specific: topic
                    obj_features1        : -anim, +count


def_no             : 6003
sense_no            : 7
def_type            : 1
lemma              : cover
entry              : cover
genus_info          : prov_superordinate1 : report
inflection          : cover covers covering covered
gram               : VB with OBJ
voice              : active
inference           : possible
subj_info           : subj_premods3      : television
                    subj3                : specific: companies
                    subj2                : specific: newspapers
                    subj1                : specific: reporters
obj_info            : obj1               : specific: event
                    obj_features1        : -anim, +count

def_no             : 6004
sense_no            : 8
def_type            : 1
lemma              : cover
entry              : cover
genus_info          : prov_superordinate1 : is enough
inflection          : cover covers covering covered
gram               : VB with OBJ
voice              : active
inference           : possible
subj_info           : subj_premods1      : sum of
                    subj1                : specific: money
                    subj_features1       : -anim
obj_info            : obj_features1      : -anim

def_no             : 6005
sense_no            : 9
def_type            : 3
gram               : COUNT N
lemma              : cover
entry              : cover
genus_info          : prov_superordinate1 : something
inflection          : cover covers

def_no             : 6007
sense_no            : 11
def_type            : 3
gram               : PLURAL N
lemma              : cover
entry              : covers
genus_info          : prov_superordinate1 : the sheet , blankets , and bedspread
precolloc_info      : bed

def_no             : 6015
sense_no            : 1
def_type            : 3
sec_gram            : UNCOUNT N
gram               : COUNT N
lemma              : covering
entry              : covering
genus_info          : prov_superordinate1 : layer of something
```

```
inflection        : covering coverings

def_no           : 6018
sense_no          : 1
def_type          : 3
gram             : COUNT N
lemma            : cover-up
entry            : cover-up
genus_info        : prov_superordinate1 : attempt
inflection        : cover-up cover-ups

def_no           : 6035
sense_no          : 3
def_type          : 1
lemma            : crack
entry            : crack
genus_info        : prov_superordinate1 : solve
inflection        : crack cracks cracking cracked
gram             : VB with OBJ
voice            : active
inference         : possible likely
subj_info         : subj_features1     : +anim, +hum
obj_info          : obj2               : specific: code
                  obj_features2      : -anim, +count
                  obj1               : specific: problem
                  obj_features1      : -anim, +count

def_no           : 6043
sense_no          : 10
def_type          : 1
lemma            : crack
entry            : have a crack at
genus_info        : prov_superordinate1 : make an attempt
inflection        : crack cracks cracking cracked
gram             : PHRASE
phrase_type       : verbal
voice            : active
inference         : possible likely
subj_info         : subj_features1     : +anim, +hum
obj_info          : obj_features1      : -anim

def_no           : 6050
sense_no          : 1
def_type          : 1
lemma            : crackle
entry            : crackle
genus_info        : prov_synonym1      : makes a series of short harsh noises
inflection        : crackle crackles crackling crackled
gram             : VB
voice            : active
inference         : possible
subj_info         : subj_features1     : -anim

def_no           : 6051
sense_no          : 1
def_type          : 3
gram             : COUNT N
lemma            : crackpot
entry            : crackpot
genus_info        : prov_superordinate1 : someone
inflection        : crackpot crackpots

def_no           : 6058
sense_no          : 1
def_type          : 3
gram             : COUNT N
lemma            : craftsman
entry            : craftsman
genus_info        : prov_superordinate1 : someone
inflection        : craftsman craftsmen

def_no           : 6072
sense_no          : 1
def_type          : 3
gram             : COUNT N
lemma            : crank
entry            : crank
genus_info        : prov_superordinate1 : someone
inflection        : crank cranks

def_no           : 6086
sense_no          : 2
def_type          : 1
lemma            : crawl
entry            : crawl
genus_info        : prov_superordinate1 : moves
inflection        : crawl crawls crawling crawled
gram             : VB
```

```
voice              : active
inference          : inherent
subj_info          : subj2              : specific: vehicle
                   subj_features2       : -anim, +count
                   subj_features1       : +anim, -hum, +animal, +count

def_no             : 6099
sense_no           : 3
def_type           : 3
gram               : MASS N
lemma              : cream
entry              : Cream
genus_info         : prov_superordinate1 : a thick substance

def_no             : 6117
sense_no           : 1
def_type           : 3
gram               : COUNT N
lemma              : creature
entry              : creature
genus_info         : prov_superordinate1 : living thing
inflection         : creature creatures

def_no             : 6141
sense_no           : 2
def_type           : 3
gram               : COUNT N
lemma              : creed
entry              : creed
genus_info         : prov_synonym1      : religion
inflection         : creed creeds

def_no             : 6452
sense_no           : 11
def_type           : 3
gram               : COUNT N
lemma              : cut
entry              : cut
genus_info         : prov_superordinate1 : reduction
inflection         : cut cuts
postcolloc_info    : colloc_prep        : preferred: in
                   colloc1              : specific: amount
                   colloc_features1     : -anim, +count

def_no             : 6700
sense_no           : 1
def_type           : 3
gram               : ADJ
lemma              : debatable
entry              : debatable
genus_info         : prov_synonym2      : not certain
                   prov_synonym1        : not definitely true
colloc_info        : colloc_postmods1   : that is
                   colloc_features1     : -anim

def_no             : 7234
sense_no           : 1
def_type           : 3
gram               : COUNT N
lemma              : device
entry              : device
genus_info         : prov_superordinate1 : object
inflection         : device devices

def_no             : 7819
sense_no           : 1
def_type           : 3
gram               : COUNT N
lemma              : document
entry              : document
genus_info         : prov_superordinate1 : piece of paper
inflection         : document documents

def_no             : 8018
sense_no           : 1
def_type           : 3
gram               : ADJ
lemma              : dramatic
entry              : dramatic
genus_info         : prov_synonym1      : sudden and noticeable
colloc_info        : colloc1            : specific: change

def_no             : 9087
sense_no           : 1
def_type           : 3
gram               : COUNT N
lemma              : event
entry              : event
```

```
genus_info        : prov_superordinate1 : something
inflection        : event events

def_no           : 9986
sense_no          : 1
def_type          : 3
gram             : ADJ
lemma            : fierce
entry            : Fierce
genus_info        : prov_synonym2      : angry
                  prov_synonym1      : aggressive
inflection        : fierce fiercer fiercest

def_no           : 10199
sense_no          : 1
def_type          : 3
gram             : COUNT N
lemma            : fish
entry            : fish
genus_info        : prov_superordinate1 : creature
inflection        : fish fishes

def_no           : 11057
sense_no          : 1
def_type          : 3
sec_gram          : UNCOUNT N
gram             : COUNT N
lemma            : fruit
entry            : fruit
genus_info        : prov_superordinate1 : something
inflection        : fruit fruits

def_no           : 12193
sense_no          : 1
def_type          : 3
gram             : COLL N
lemma            : group
entry            : group
genus_info        : prov_superordinate2 : number of things
                  prov_superordinate1 : number of people

def_no           : 13377
sense_no          : 1
def_type          : 3
gram             : COUNT N
lemma            : house
entry            : house
genus_info        : prov_superordinate1 : building
inflection        : house houses

def_no           : 14196
sense_no          : 1
def_type          : 3
gram             : ADJ
lemma            : iniquitous
entry            : Iniquitous
genus_info        : prov_superordinate1 : bad and unfair

def_no           : 14347
sense_no          : 1
def_type          : 3
gram             : COUNT N
lemma            : instrument
entry            : instrument
genus_info        : prov_superordinate2 : device
                  prov_superordinate1 : tool
                  purpose1          : do a particular task
inflection        : instrument instruments

def_no           : 15219
sense_no          : 2
def_type          : 3
gram             : ADJ
lemma            : lame
entry            : lame
genus_info        : prov_synonym2      : weak
                  prov_synonym1      : poor
colloc_info       : colloc3           : specific : remark
                  colloc2           : specific : argument
                  colloc1           : specific : excuse

def_no           : 16000
sense_no          : 1
def_type          : 3
gram             : ADJ
lemma            : livid
entry            : livid
genus_info        : prov_superordinate1 : angry
```

```
colloc_info       : colloc_postmods1   : who is
                  colloc_features1   : +anim, +hum

def_no            : 16390
sense_no          : 1
def_type          : 3
gram              : COUNT N
lemma             : machine
entry             : machine
genus_info        : prov_superordinate1 : piece of equipment
inflection        : machine machines

def_no            : 16928
sense_no          : 1
def_type          : 3
gram              : COUNT N
lemma             : meeting
entry             : meeting
genus_info        : prov_superordinate1 : event
inflection        : meeting meetings

def_no            : 18449
sense_no          : 1
def_type          : 3
gram              : COUNT N
lemma             : object
entry             : object
genus_info        : prov_superordinate1 : anything
inflection        : object objects

def_no            : 18623
sense_no          : 4
def_type          : 3
gram              : COUNT N
lemma             : official
entry             : official
genus_info        : prov_superordinate1 : person
inflection        : official officials

def_no            : 18940
sense_no          : 1
def_type          : 3
gram              : COUNT N
lemma             : organization
entry             : organization
genus_info        : prov_superordinate1 : a group of people
inflection        : organization organizations

def_no            : 19759
sense_no          : 1
def_type          : 3
gram              : ADJ
lemma             : penitent
entry             : penitent
genus_info        : prov_superordinate1 : sorry
colloc_info       : colloc_postmods1   : who is
                  colloc_features1   : +anim, +hum

def_no            : 19820
sense_no          : 1
def_type          : 3
gram              : COUNT N
lemma             : period
entry             : period
genus_info        : prov_superordinate1 : length of time
inflection        : period periods
precolloc_info    : particular

def_no            : 20005
sense_no          : 5
def_type          : 1
lemma             : pick
entry             : pick
genus_info        : prov_superordinate1 : open
inflection        : pick picks picking picked
gram              : VB with OBJ
voice             : active
inference         : possible negative/unlikely
subj_info         : subj_features1     : +anim, +hum
obj_info          : obj1               : specific: lock
                  obj_features1      : -anim, +count

def_no            : 20007
sense_no          : 5
def_type          : 1
lemma             : pick
entry             : pick at
genus_info        : prov_superordinate1 : eat
```

```
inflection        : pick picks picking picked
gram              : PHR VB
voice             : active
inference         : possible likely
subj_info         : subj_features1     : +anim, +hum
obj_info          : obj_premods1       : food
                  obj1               : specific: you are eating
                  obj_features1      : -anim

def_no            : 20013
sense_no          : 4
def_type          : 1
lemma             : pick
entry             : pick up
genus_info        : prov_superordinate1 : acquire
inflection        : pick picks picking picked
gram              : PHR VB
voice             : active
inference         : possible likely
subj_info         : subj_features1     : +anim, +hum
obj_info          : obj2               : specific: idea
                  obj_features2      : -anim, +count
                  obj1               : specific: skill
                  obj_features1      : -anim, +count

def_no            : 20023
sense_no          : 1
def_type          : 3
gram              : COUNT N
lemma             : pictorial
entry             : Pictorial
genus_info        : prov_synonym2      : using pictures
                  prov_synonym1      : relating to
inflection        : pictorial

def_no            : 20024
sense_no          : 1
def_type          : 3
gram              : COUNT N
lemma             : picture
entry             : picture
genus_info        : prov_synonym2      : painting
                  prov_synonym1      : drawing
inflection        : picture pictures

def_no            : 20026
sense_no          : 3
def_type          : 1
lemma             : picture
entry             : picture
genus_info        : prov_superordinate1 : appear
inflection        : picture pictures picturing pictured
gram              : VB with OBJ
voice             : passive
inference         : possible
obj_info          : obj_features2      : -anim
                  obj_features1      : +anim, +hum
obliq_info        : obliq_prep         : preferred: in
                  obliq2             : specific: magazine
                  obliq_features2    : -anim, +count
                  obliq1             : specific: newspaper
                  obliq_features1    : -anim, +count

def_no            : 20036
sense_no          : 1
def_type          : 3
gram              : COUNT N
lemma             : piece
entry             : piece
genus_info        : prov_synonym3      : section of
                  prov_synonym2      : part
                  prov_synonym1      : portion
inflection        : piece pieces
postcolloc_info   : colloc_prep        : preferred: of
                  colloc_features1   : -anim

def_no            : 20037
sense_no          : 2
def_type          : 3
gram              : COUNT N
lemma             : piece
entry             : piece
genus_info        : prov_superordinate1 : item of
inflection        : piece pieces
postcolloc_info   : colloc_prep        : preferred: of
                  colloc_features1   : -anim
```

```
def_no          : 20039
sense_no        : 4
def_type        : 3
gram            : COUNT N
lemma           : piece
entry           : piece
genus_info      : prov_superordinate1 : something
inflection      : piece pieces

def_no          : 20062
sense_no        : 1
def_type        : 3
sec_gram        : UNCOUNT N
gram            : COUNT N
lemma           : pigment
entry           : pigment
genus_info      : prov_superordinate1 : substance
inflection      : pigment pigments

def_no          : 20063
sense_no        : 1
def_type        : 3
gram            : COUNT N
lemma           : pigsty
entry           : pigsty
genus_info      : prov_superordinate1 : hut
inflection      : pigsty pigsties

def_no          : 20064
sense_no        : 1
def_type        : 3
gram            : COUNT N
lemma           : pigtail
entry           : pigtail
genus_info      : prov_synonym1       : length of plaited hair
inflection      : pigtail pigtails

def_no          : 20079
sense_no        : 2
def_type        : 3
gram            : SING N
lemma           : pill
entry           : pill
genus_info      : prov_superordinate1 : type of drug

def_no          : 20086
sense_no        : 1
def_type        : 3
gram            : COUNT N
lemma           : pilot
entry           : pilot
genus_info      : prov_superordinate1 : person
inflection      : pilot pilots

def_no          : 20153
sense_no        : 1
def_type        : 3
gram            : COUNT N
lemma           : piston
entry           : piston
genus_info      : prov_superordinate2 : metal disc
                  prov_superordinate1 : cylinder
inflection      : piston pistons

def_no          : 20154
sense_no        : 1
def_type        : 3
gram            : COUNT N
lemma           : pit
entry           : pit
genus_info      : prov_superordinate1 : hole
inflection      : pit pits

def_no          : 20171
sense_no        : 1
def_type        : 3
gram            : COUNT N
lemma           : pitcher
entry           : pitcher
genus_info      : prov_synonym1       : jug
inflection      : pitcher pitchers

def_no          : 20190
sense_no        : 1
def_type        : 3
gram            : COUNT N
lemma           : placard
```

```
entry           : placard
genus_info       : prov_superordinate1 : notice
inflection       : placard placards

def_no          : 20192
sense_no         : 1
def_type         : 3
gram            : COUNT N
lemma           : place
entry           : place
genus_info       : prov_synonym5      : country
                  prov_synonym4      : town
                  prov_synonym3      : area
                  prov_synonym2      : building
                  prov_synonym1      : point
inflection       : place places

def_no          : 20842
sense_no         : 2
def_type         : 3
gram            : ADJ
lemma           : precise
entry           : precise
genus_info       : prov_synonym1      : exact and accurate
colloc_info      : colloc_postmods1   : that is
                  colloc_features1   : -anim

def_no          : 23142
sense_no         : 1
def_type         : 3
gram            : ADJ
lemma           : robust
entry           : robust
genus_info       : prov_synonym1      : strong and healthy
colloc_info      : colloc_postmods2   : that is
                  colloc_features2   : -anim
                  colloc_features1   : +anim, +hum

def_no          : 23203
sense_no         : 1
def_type         : 3
gram            : COUNT N
lemma           : room
entry           : room
genus_info       : prov_superordinate1 : one of the separate sections
inflection       : room rooms

def_no          : 24481
sense_no         : 2
def_type         : 3
gram            : COUNT N
lemma           : shape
entry           : shape
genus_info       : prov_superordinate1 : something
inflection       : shape shapes

def_no          : 24647
sense_no         : 1
def_type         : 3
gram            : COUNT N
lemma           : shop
entry           : shop
genus_info       : prov_superordinate2 : part of a building
                  prov_superordinate1 : building
inflection       : shop shops

def_no          : 25862
sense_no         : 1
def_type         : 3
gram            : ADJ
lemma           : spindly
entry           : spindly
genus_info       : prov_synonym1      : long , thin , and weak
colloc_info      : colloc_postmods1   : that is
                  colloc_features1   : -anim

def_no          : 26285
sense_no         : 1
def_type         : 3
gram            : COUNT N
lemma           : statement
entry           : statement
genus_info       : prov_superordinate1 : something
inflection       : statement statements

def_no          : 26335
sense_no         : 5
def_type         : 3
```

```
gram              : ADJ
lemma             : steady
entry             : steady
genus_info         : prov_synonym1      : sensible and reliable
inflection         : steady steadier steadiest
colloc_info        : colloc_features1    : +anim, +hum, +count

def_no            : 26829
sense_no           : 2
def_type           : 3
gram              : COUNT N
lemma             : structure
entry             : structure
genus_info         : prov_superordinate1 : something
inflection         : structure structures

def_no            : 26980
sense_no           : 1
def_type           : 3
gram              : COUNT N
lemma             : substance
entry             : substance
genus_info         : prov_synonym3      : liquid
                   prov_synonym2      : powder
                   prov_synonym1      : solid
inflection         : substance substances

def_no            : 27097
sense_no           : 1
def_type           : 3
gram              : COUNT N
lemma             : sum
entry             : sum
genus_info         : prov_synonym1      : amount of
inflection         : sum sums
postcolloc_info     : colloc_prep        : preferred: of
                   colloc1            : specific: money
                   colloc_features1    : -anim

def_no            : 27098
sense_no           : 2
def_type           : 3
gram              : COUNT N
lemma             : sum
entry             : sum
genus_info         : prov_superordinate1 : calculation
inflection         : sum sums

def_no            : 27853
sense_no           : 3
def_type           : 3
gram              : ADJ
lemma             : tender
entry             : Tender
genus_info         : prov_superordinate1 : soft and easy
inflection         : tender tenderest tenders
colloc_info        : colloc1            : specific: meat

def_no            : 28801
sense_no           : 1
def_type           : 3
gram              : COUNT N
lemma             : tree
entry             : tree
genus_info         : prov_superordinate1 : plant
inflection         : tree trees

def_no            : 29894
sense_no           : 1
def_type           : 3
gram              : COUNT N
lemma             : vegetable
entry             : Vegetables
genus_info         : prov_superordinate1 : plants
inflection         : vegetable vegetables
form              : plural

def_no            : 29901
sense_no           : 1
def_type           : 3
gram              : COUNT N
lemma             : vehicle
entry             : vehicle
genus_info         : prov_superordinate1 : machine
inflection         : vehicle vehicles
```