# Semantic networks for automatic coding

*Leon Willenborg*

**Discussion paper (201213)**

Statistics Netherlands

The Hague/Heerlen, 2012

## Explanation of symbols

| | |
|---|---|
| **.** | data not available |
| ***** | provisional figure |
| ****** | revised provisional figure (but not definite) |
| **x** | publication prohibited (confidential figure) |
| **–** | nil |
| **–** | (between two figures) inclusive |
| **0 (0.0)** | less than half of unit concerned |
| **empty cell** | not applicable |
| **2011–2012** | 2011 to 2012 inclusive |
| **2011/2012** | average for 2011 up to and including 2012 |
| **2011/’12** | crop year, financial year, school year etc. beginning in 2011 and ending in 2012 |
| **2009/’10–2011/’12** | crop year, financial year, etc. 2009/’10 to 2011/’12 inclusive |

Due to rounding, some totals may not correspond with the sum of the separate figures.

**Publisher**
Statistics Netherlands
Henri Faasdreef 312
2492 JP The Hague

**Prepress**
Statistics Netherlands
Grafimedia

**Cover**
Teldesign, Rotterdam

**Information**
Telephone +31 88 570 70 70
Telefax +31 70 337 59 94
Via contact form:
www.cbs.nl/information

**Where to order**
E-mail: verkoop@cbs.nl
Telefax +31 45 570 62 68

**Internet**
www.cbs.nl

ISSN: 1572-0314

60083201213 X-10

# Semantic networks for automatic coding

Leon Willenborg

*Summary: In this paper a methodology for automatic coding is suggested, which is fairly general. Characteristic for this approach  is that each code (say for a business activity) is characterized by one or more combinations of code words, or C-words. These combinations of C-words can be seen as definitions of the various codes used. It is assumed that the order of the C-words is irrelevant to describe a code. Because it is unlikely that people will use exactly those C-words when describing a business activity, synonyms, hyponyms and hyperonyms for the C-words are needed as well. These words connect the definition of the codes in the classification used on the one hand and the descriptions provided by respondent on the other. These words are called D-words. A semantic network is used to provide a bridge between the descriptions and the codes.*

.

*Key words: text interpretation, natural language processing, computational linguistics, derivation, code assignment, classification, knowledge representation, semantic network.*

# 1. Introduction

## 1.1 Background

This paper describes a methodology for automatic coding that has its origins in a project at Statistics Netherlands that was carried out in 2004. The aim was the development of an interactive tool that could be used at the Chambers of Commerce in The Netherlands to elicit detailed information about businesses and their activities. The input for this tool was supposed to be descriptions of the activities. Businesses were supposed to provide the descriptions, phrasing it in the way they preferred. A desk clerk should input the information into the system, using this tool. It was not assumed that the respondents had any knowledge of the business classification that was used to code the information provided. The project delivered a prototype tool that was later upgraded to a coding tool that has been used by the Chambers of Commerce ever since.

The current paper describes ideas that have been used in the coding tool. The description, however, has been made more general, in the sense that it does not only apply to business activities. It also contains ideas that have evolved and matured since the project was finished.

## 1.2 Coding

Coding in official statistics is a process of deducing a variable. A deduced variable is a variable whose value for a sample entity is a function of the values of some other, auxiliary, variables that have been observed. Typically, the auxiliary variables are categorical variables. What sets coding apart is that, at least some of the auxiliary variables are text variables. What makes coding difficult is the fact that the auxiliary variables contain free text, that is, phrases formulated freely by the respondents. So to deduce the value of the coded variable (usually an element of a classification) there is the problem of understanding ‘written text’ or ‘written language’. The derived variable typically takes values in a classification, which is usually representable by a tree.

So coding is the process of interpreting descriptions given in writing (of a business activity, an occupation, a disease, etc) with respect to a suitable classification. The information is usually provided by respondents who are not supposed to have any knowledge about the classification that is used to interpret their response. This response is typically about the education one received, the job title, the goods being produced, repaired, transported, etc, the services provided, the business activity of a company, diseases that persons (or animals) can be suffering from, the causes of death of persons, etc. These variables have in common that the corresponding domains are big, and that a rather complicated tree structure in the form of a classification tree is defined on them. It requires expertise of the various classifications, and in particular the classifying principles underlying them, to provide the correct answer for each respondent. A typical respondent cannot be expected to possess such knowledge. So the solution is that the respondent provides an answer to such a question in his own words, and this answer is interpreted by either a coding expert or a coding program, or both. Instead of using a single question, it is possible to use several questions, each pertaining to a specific aspect and each of them relatively simple to answer. Some of these questions could even be closed, that is, with an explicitly given set of possible answers.

Coding can also be seen as a translation problem, translating descriptions to codes. In automatic coding this translation is carried out by computer, completely or partially. If it is done completely, the user information is coded without asking for feedback information from the person who provided the information. If it is done partly, the coding system needs this feedback to make certain decisions that lead to a code (or a decision that no such code can be generated on the basis of the information provided). Coding systems that find a code fully automatically are (currently) an illusion, and maybe always will be. One problem is that the descriptions that are fed into the coding system are not ideal, but may contain insufficient information to lead to a unique code. In practice, some form of feedback may be needed, if possible, for instance in case the information is not detailed enough, or ambiguous, or unintelligible, etc. In case no such feedback is possible, one should allow for the possibility that the information provided is insufficient to decide for a single code (at the level of precision desired). One can then either settle for a less precise code or for no code at all (if there is too much ambiguity) and hope that by manual coding a code can be attached to the description provided.

Coding can also be seen as a classification problem (with a twist), as studied in areas such as data mining. D-words in the descriptions provided by respondents are used as predictors of the unknown codes. What distinguishes the automatic coding problem from the numerical prediction problems is that the prediction variables are alphanumeric and not numeric. The D-words in a description are drawn from some specialized vocabulary, geared at a particular application that can be fairly extensive. Because the predictor variable consists of words rather than of numerical values, one is faced here with some language problems, which are absent from standard applications in this area. These problems are due to incorrect spelling of words (problems with orthography), the use of specialized vocabulary, jargon, dialect, etc. (which requires to build a reasonably complete vocabulary), dealing with synonyms, hyperonyms and hyponyms (which relates to a thesaurus function on top of the vocabulary). To cater for all these aspects may require quite some effort.

Coding can be carried out under various circumstances: there is the traditional ‘manual coding‘ which involves coding of written descriptions on paper questionnaires. This is done by ‘coders’ or, equivalently, ‘coding experts’, i.e. persons. Next in the list is semi-automatic coding, where the coding decisions are still being taken by coders, but this time they are supported by a simple information system, e.g. an electronic filing system with auxiliary information. Finally, at the top of the list, there is the automatic coding system, that we just briefly discussed. This will be the main focus of the paper, that is, using semantic networks for automatic coding.

## 1.3 Automatic coding

In this section we describe the typical setting for a coding process in which the method that we propose in this paper (i.e. using a semantic network) is supposed to work. It concerns the assumptions made with respect to the respondents in particular concerning the classification used by the statistical office, the type of (initial) response expected, what to do in case feedback is possible or not, etc. It also tries to motivate why coding is used at all, and what kind of problems one faces in coding, and also how these are supposed to be dealt with.

The idea is that respondents provide written descriptions pertaining to the subject of the coding problem. Answers of this type are called answers to 'open questions', that is, questions where respondents can formulate their answers in their own words. This increased flexibility has a drawback: it puts the burden of interpreting the answers to such questions to the statistical institute. But sometimes this situation is preferable. In case of coding, the problem is often a complex and specialised classification that is being used by the statistical institute, and respondents cannot be supposed to have knowledge of this classification. So it is better if they try to explain in their own words what they have to say about the subject. The subjects where coding is applied typically are 'occupation', 'education', 'business activity', 'goods' (produced, traded, transported, used, etc.), 'diseases', 'causes of death', to mention but a few.

On the basis of descriptions provided by respondents an automatic coding system attempts to classify the items described with respect to the given classification, using a computer program to this purpose. In case a description is adequate the coding system should come up with an answer, i.e. a code from the classification being used. In case it is not detailed enough or confusing the program should respond accordingly. Then it depends on the situation at hand what should happen next. In case the respondent is still available he should be asked to supply additional information so that a unique code can be found in a next try. This interaction may be repeated several times, either until a unique answer (code) is found or the search is given up. In another situation where a respondent is not available to correct or supplement his answer, a human coding expert (a coder, for short) may be called in to look at the description provided, and possible additional material, and try to deduce (or guess) an answer (a code) from this.

## 1.4 Overview

The current paper describes a semantic network for automatic coding and its employment. The architecture of such a semantic network is independent of an area of application. This implies that a general shell could be used to handle descriptions (spell checking & correcting, parsing, and other purely language related tasks), to make inferences using the semantic network to find codes corresponding with descriptions, and to handle the feedback with the reader in case not a single code cannot be obtained. The semantic network is the core of such a system, and the idea is that it is also separate from the 'handling' software and can easily be replaced by another such semantic network for another application. The semantic network is as independent data for the handling software.

We briefly describe how the automatic coding process based using a semantic network as described in the present paper is supposed to operate. We start with the basic input, which is a description, provided by a respondent. This is read and parsed by the system and the individual tokens (words) are identified. Possibly the description is first spell checked and corrected.

The words obtained from the description are then matched with the set of D-words in the semantic network. If all words in the description are recognized as being part of this set, we are ready for the next phase. If not, efforts have to be developed to handle the unrecognized words in the description. They have to be added to the semantic network. Coding experts have to look at these words and decide to add them to the semantic network, or rather to incorporate them. This is a relatively laborious job, but an essential one, to let the semantic network 'learn'. After the semantic network has been extended, the words that were originally not recognized (as they were not in the set of D-words) now should be recognized.

The next step seeks to replace the D-words by so-called C-words. They form so to speak a subset of the D-words and they are used to describe the codes in the classification system that the coding system uses. This reduction to C-words is via synonyms and hyperonyms, semantic relations that denote semantic equivalence or generalization, respectively. The codes in the classification are described in terms of combinations of concepts. Each concept is a set of C-words. In the ideal case one code is now found in the coding system, at the desired level of detail. This code is assigned to the description. In case no such code is found, the information in the description is imperfect: either no code is found or more than one code is found (at the right level of detail). In both cases feedback from the respondent is needed in order to arrive at a single code. The description may contain too much or too little information. So this requires a setting were the feedback can indeed be delivered.

To understand the special words in which the model is described (D-word, C-word, homonym, hyperonym, etc) the reader is referred to appendix D.

## 1.5 Structure of the paper

The remainder of the paper is structured as follows. In successive chapters various ingredients of a semantic network for coding are discussed. We start in Chapter 2 with discussing words, divided into two groups: C-words and D-words. In Chapter 3 various constructs made from specific types of words are considered, namely concepts, codes and classifications. Then, in Chapter 4, the various kinds of relations that exist between words, concepts and codes (from classifications) are discussed. Chapter 5 is entirely devoted to illustrating aspects of the approach set out in previous chapters by presenting a few examples.. In Chapter 6 we discuss briefly, how an automatic coding process works using a semantic network could work. We start with the ‘happy flow’, which deals with the ideal case. This gives an insight in the coding process under ideal conditions. In practice many exceptions can occur to the ideal case. We then continue discussing these in brief. Full treatment would require quite a lot of extra space, possibly at book length. In Chapter 7 a summary and discussion of the main points of the approach taken in this paper are given. The paper is completed with a list of references and four appendixes, the last one of which is a glossary.

# 2. Words

The basic building blocks for a semantic network for coding are words. We consider two classes of words: C-words that are used to define concepts in a classification. These concepts in turn are used to define codes, or, equivalently, nodes in a classification. They form the target part of a coding activity. At the other end, we have the D-words, that is, words that actually appear in descriptions given by respondents. The D-words and C-words may be related through semantic relationships. D-words not connected to C-words either are not useful, as they carry no or little information, or they are new and have not been incorporated into the semantic network.

Below we go into each of these sets of words and so some way to explain their meaning.

## 2.1 C-words: building blocks for concepts

C-words are the building blocks of the coding methodology described in the present paper. What can be build with them are what in the present paper are called concepts. Concepts are used to define codes. A *concept* is represented as a finite set of C-words. This implies that the order of the C-words used to describe a concept is unimportant. It should be stressed that this is an assumption not an established fact.

The C-words have to be considered as formal words used to define the codes in a formal way. As a matter of fact, it is not even important from which language(s) the C-words are taken. Their semantic content is important, not the (natural) language(s) from which they are taken. Of course, the idea is that one language should be chosen (and specified) and all C-words are supposed to be taken from this language. However, it would be possible to take C-words from a language different from that in which the descriptions are formulated. This is convenient in countries with more than one official language, as it would allow to define the code structure in one language and add appropriate D-words in the other language, as synonyms, hyponyms or hyperonyms, at the correct places.

The C-words should be chosen in such a way that they can describe all codes in the classification, including the most detailed ones that will be used, adequately, via concepts. We assume that the most detailed level defines the desired level of coding. It is therefore not necessary to consider more detailed codes. Less detailed codes however cannot be discarded since each higher level code consists of two or more codes. So we exclude the possibility that a node that is not a terminal node has exactly one child-node. In such a case there would be no branching, and such a node could be left out, without losing any information. (This ‘anomaly’ actually does occur in practice!)

This does not imply that respondents cannot give more detailed answers than the coding level requests. But through the use of suitable semantic relationships this information can be channelled to the desired level.

For a given classification the set of C-words to describe its concepts can be viewed as a fixed set. If the classification changes, it may be necessary to modify the corresponding set of C-words accordingly.

### 2.2 D-words: building blocks for descriptions

D-words are the words that are actually found in descriptions provided by respondents for the topic of the classification. In principle there are two options here:

1. correctly and incorrectly spelled words. In this case the raw input of the descriptions is directly used, without any modifications.
2. only correctly spelled words. In this case it is assumed that all descriptions are first spell-checked and corrected.

In the first case the set of D-words is bigger than in the second case. The advantage of this set is that it can be fed with the words as they appear in strings. This can occur any time new descriptions are received. But these ‘raw’ words, if not already in the currently existing set of D-words, have to be interpreted and classified (synonyms, etc have to be specified). In the second case the polishing step (the spell checking and correcting) needed to obtain the words may potentially destroy useful information, assuming that it is done automatically. If it is done interactively, it may take a similar effort as in the first case when dealing with new D-words, not already in the current set of D-words. Some experimentation is probably needed to find out which option to choose for the set of D-words.

Contrary to the set of C-words, the set of D-words is to be considered open-ended, as new words may pop up as long as new empirical material (i.e. descriptions) is collected. In case this happens, such a word should be integrated into the semantic network, by using semantic relationships. This means that it should be connected to the C-words via semantic relationships. These have to be defined by workers maintaining the semantic network used. This can be considered as a learning activity of the semantic network.

At any time the set of C-words is a subset of the set of corresponding D-words. In fact, the C-words associated with a classification can be considered as the starting point for the set of D-words, even without any empirical material (i.e. descriptions).

The descriptions are the sources of additional D-words. Descriptions are parsed into words and any word that is not on the list, i.e. in the current set of D-words, is added.

# 3. Concepts, codes and classifications

In this chapter we consider fundamental building blocks for a semantic network used for automatic coding, such as concepts, codes and classifications.

## 3.1 Concepts: building blocks for codes

As we have indicated before concepts are defined in terms of C-words. More precisely we assume that each concept is represented by a finite set of C-words. This implies that the order of the C-words comprising a concept is irrelevant. This is an assumption, which probably holds for the majority of concepts in the majority applications. It remains to be seen if this leads to problems in some cases.[1]

In order to be consistent there are some constraints applicable to the set of concepts. For instance, if A and B are concepts characterizing the same code c, A should not be contained in B (i.e. $A \not\subset B$) , and B not in A (i.e. $B \not\subset A$). Also we have that for each concept there is a code. Furthermore, a concept can correspond to only one code. This assumes that we do not have concepts that are synonymous and belong to different codes.

It can a priori not be excluded that C-words are synonyms, it should be avoided that the concepts such words are involved in are different. Say a and b are C-words and synonyms, and c is another C-word. Then the concepts {a,c} and {b,c} can not be associated to different codes, because the concepts are synonymous. If d is another C-word and not synonymous to a, b or c then {a,c} and {b,d} can be associated with different codes, as {a,c} and {a,d} are not synonymous.

Both these conditions imply that there exists a map $\kappa : K \to C$ from $K$ the set of concepts (for a particular application) to the set $C$ of codes (in the classification for that application), and this map $\kappa$ is surjective. It induces an equivalence relation ~ on $K$ : For $K_1, K_2 \in K$ we have $K_1 \sim K_2$ if (and only if) $\kappa(K_1) = \kappa(K_2)$ . The equivalence classes consist of descriptions that correspond to the same code. The set of equivalence classes of K induced by $\kappa$ is denoted by $K / \kappa$ . In plain words it simply means that the set of concepts can be partitioned in such a way that each part contains all the concepts used to characterize a particular code.

[1] In the following example (due to Sander Scholtus) the order of the words would be important: 'selling of plant and flower seeds' and 'planting of seeds and selling flowers'. This would in borh cases (probably) lead to the set of C-words {sell, plant, flower, seed} and which would be ambiguous.

## 3.2 Codes: building blocks for classifications

We assume that for our coding problem a classification is given. This can be about goods, business activities, occupation, education, diseases, causes of death, etc. We assume this classification to be a directed tree, where the direction of each arc is from special to more general. Then it holds that any two nodes are connected by a unique shortest path. Usually there is also a unique root node, which can be viewed as the origin of the classification. See Figure 2.

With each node of this directed tree structure we assume that a code has been associated. Such a code describes a particular good, business activity, occupation, type of education, cause of death, etc. that is associated with the corresponding node.

**Figure 1. D-words, C-words and stop words**

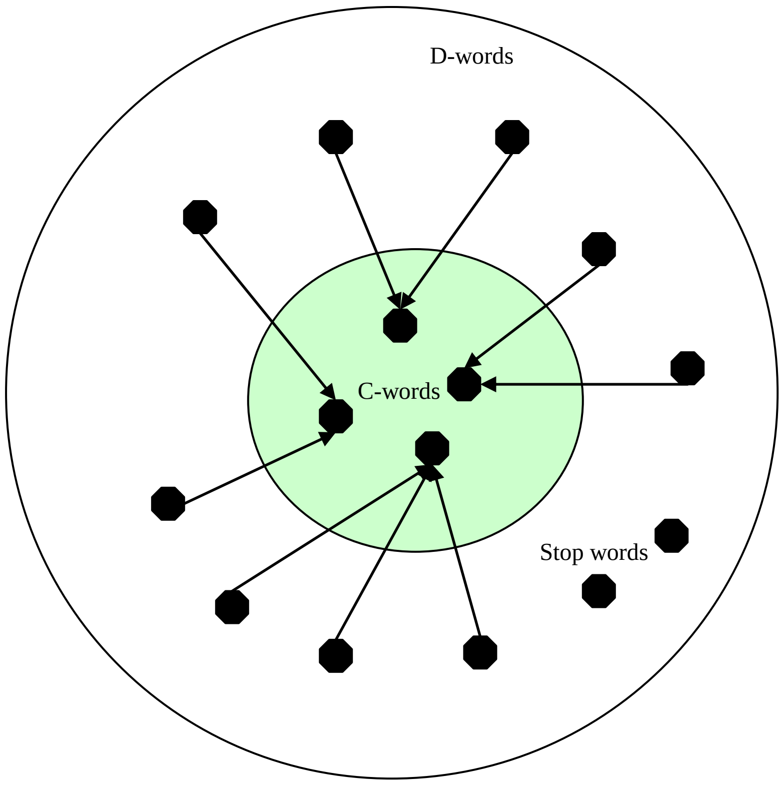
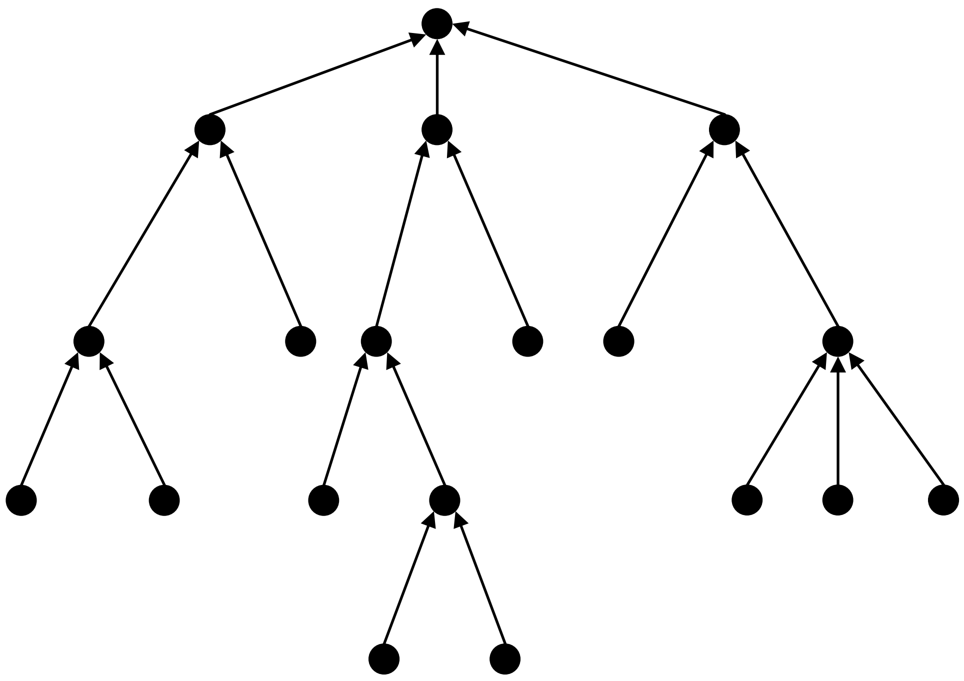


**Figure 2. A classification as a directed tree**



In the previous section it was indicated that each code is characterized by a set of concepts, and each concept is defined by a set of C-words. Just above we stated that a code is associated with a classification. It should fit into the hierarchy defined by the classification. A code is so to speak part of two structures, one derived from language (through the C-words and the semantic structure defined on this set) and the other from the classification being used (which we consider a given structure).

In the next section these relationships are considered more closely. These points are illustrated in Figures 3 and 4, respectively.

In Figure 3 the nodes are labelled in a way consistent with the hierarchy. Also indicated (in grey) are the nodes that are associated with the target codes. The target codes for the automatic coding exercise define the set of codes that should result from the coding process, if all goes according to plan.

In Figure 4 a code C is shown and four nodes corresponding to concepts that define this code. Each concept is defined in terms of C-words. Figure 4 should be read as follows

$$C = (a \wedge b \wedge c) \vee (a \wedge e) \vee (b \wedge c \wedge e) \vee (d \wedge e)\,. \qquad (3.1)$$

This means that C is the generated code if the concepts $\{a, bc\}$ , $\{a, e\}$ , $\{b, c, e\}$ or $(d, e\}$ are activated. A concept is activated if and only if the C-words of which it exists are activated. So the concept $\{a, b, c\}$ is activated if and only if the C-words a, b and c are activated. Something similar holds for the other three concepts. Note the

**Figure 3. Classification tree, codes and target codes (in grey).**

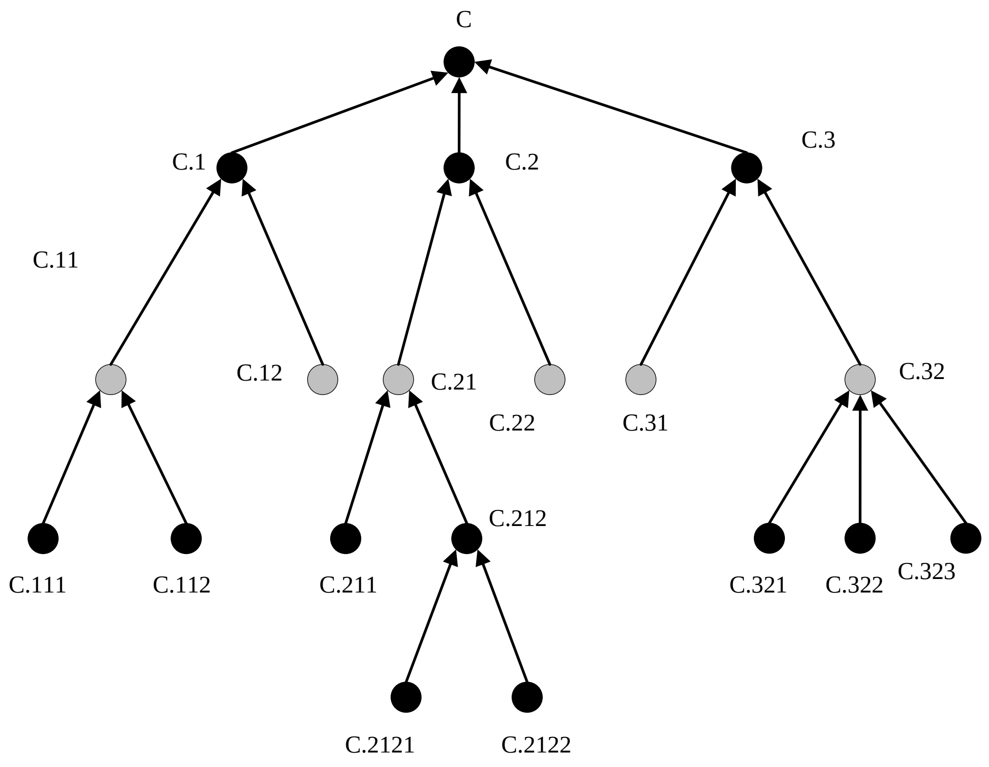
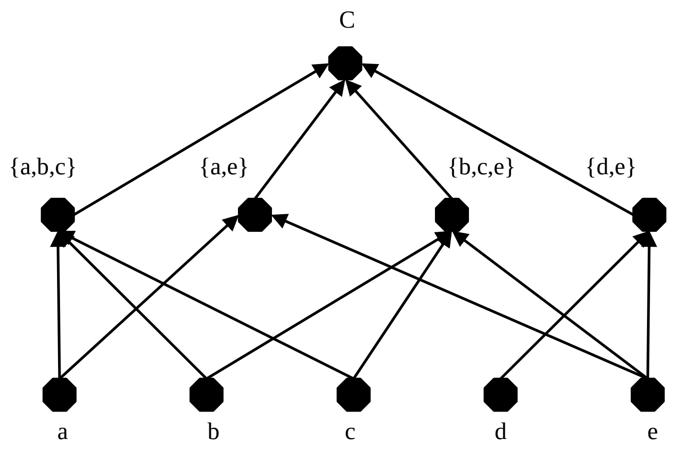


**Figure 4. A code defined in terms of concepts, which in turn are defined in terms of C-words.**



peculiarity of the interpretation of Figure 4 (and similar such digraphs). For the concepts we need conjunctions ($\wedge$) and for the code description we need disjunctions ($\vee$). So the idea is that code C is generated if one of the defining concepts is activated. This happens when all the C-words of this concept are activated by D-words generated from in a description. (3.1) viewed as a Boolean expression, it is said to be in disjunctive normal form.

**Remark 3.1** Defining codes in terms of one or more combinations of concepts would not go beyond the framework sketched. The reason is because any Boolean expression can be rewritten in disjunctive normal form. The conjunctive terms should be defined as concepts. Then we would also yield a code defined as a disjunction of concepts. ■

So once all codes in the classification have been described in terms of concepts, the next thing to do is to use the structure of the classification as a highway to connect the various codes, each defined in terms of concepts. It should be checked if, by using semantic relations (i.c. hyponymy) and by removing C-words from concepts, one can go from any node / code in the classification (except the top node) to its (unique) parent code / node.

We now briefly consider special codes that are often used in practice, because they are so convenient. Such codes define a remainder category, applying to all the cases that do not fall under the heading of the alternative possibilities. The label for such a category could formally be something like ‘other’ or ‘none of the above’ (e.g. in a questionnaire, where it is the last-mentioned category of a question). Of course, as a description of the category for our purposes it is not useful, although this code is well-defined. It is a negative definition of a code. See Figure 5 as an example of this.

Codes with exceptions in their characterization need to be handled differently in the coding process from codes without exceptions. It is a somewhat specialized topic so it is not discussed in the main text, but in Appendix B.

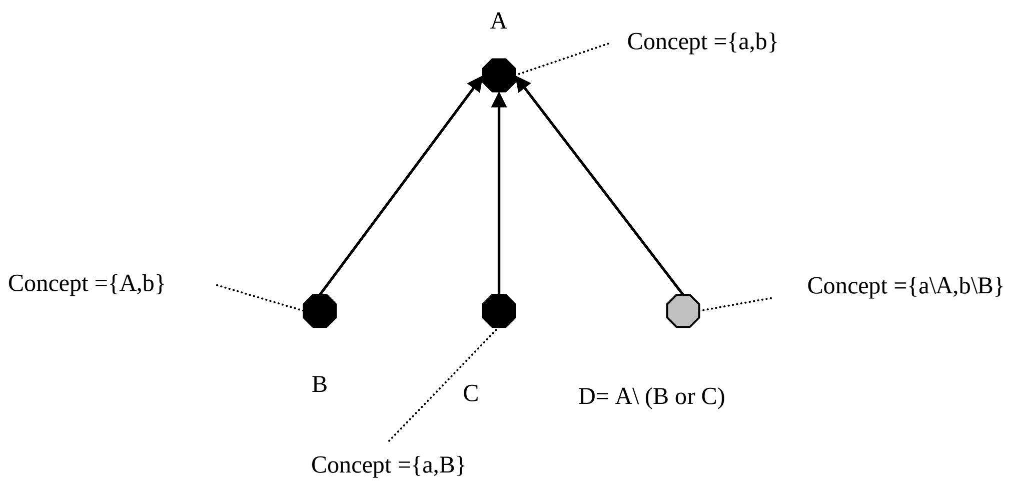
From practical experience (e.g. earlier held surveys) it is known that certain codes occur more frequently in a population than others. This kind of knowledge can be incorporated into the data by using weights for each code, reflecting the appearance of the codes in practice. Such a weight is (proportional to) an observed frequency.

These weights (or frequencies) may help to decide to choose a code when a description leads to various alternatives.

The methodology is partly language independent and partly language dependent, as we remarked already. In the present section we consider some language dependent issues. Although not belonging to the core of the methodology, it is nevertheless important when it is applied in practice. Any implementation has to deal with these issues.

**Remark 3.2** We assume that the domain of the variable being coded is the set consisting of all its codes. Usually there is a hierarchy defined on this domain (which is a digraph), which signifies a relationship like hyperonymy/hyponymy for words. But the exact nature of this relationship we can take for granted.This structure is usually a directed tree structure, yielding a partial order by taking the transitive closure of the hierarchy. But, more generally, it could also be a directed acyclic graph (DAG). See Appendix A. ■

**Figure 5. Exception code (in grey).**



When a classification is used in a coding application, a desired level of coding should be specified. Or if the classification is not organized in this way, individual nodes should be marked as at the desired level of detail. See Figure 6. This means that more detailed results are replaced by less detailed ones, at the corresponding desired level of detail. This can be done automatically by following the arcs in the classification tree until the node is reached at the right level. In case a description is not detailed enough and there are more nodes at the desired level that are more detailed, the answer provided is not precise enough. Additional information is needed to single out one of these nodes at the target level.

**Remark 3.3** When building the semantic network it seems natural to start with the codes and describe these via concepts using C-words. This set of C-words can also be taken as the nucleus of the set of D-words. One can then continue with looking at the semantic structure at the set of C-words. Then the set of C-words can be expanded by adding words taken from descriptions, and from one's knowledge of the language in which the descriptions are stated. These new D-words also need to be put in the semantic framework started with the C-words. So it is an approach from the back to the front, so to speak. ■

## 4. Relations and constraints

In the approach to coding described in this paper, a semantic network is the centrepiece. It establishes the link between descriptions on the one hand and the codes in the classification on the other. The link is established using connections among and between D- and C-words, concepts and codes. Various relations that exist among and between the fundamental objects (words, concepts, codes, classification) discussed in the previous section, are detailed in the present one.

The semantic relations that exist at the word level imply relations at the concept and code level. The codes are part of a classification which is a tree structure. This structure should be compatible with the relations implied by those at code level via the relations that hold at the concept level.

The aim of this section is to indicate the various relationships that hold between entities in a semantic network for coding. And also to point out that for a semantic network certain constraints need to hold in order for it to be consistent. These are not automatically satisfied and have to be explicitly checked for such a semantic network. This checking needs to be repeated every time it is updated as a result of learning.

## 4.1 Semantic relations among words

As we have explained before, the building blocks for concepts (and therefore, indirectly, also for codes) are C-words. To make the coding possible we need to link D-words to C-words, using semantic relationships. Important semantic relationships that will be used for this are presented in Table 1.

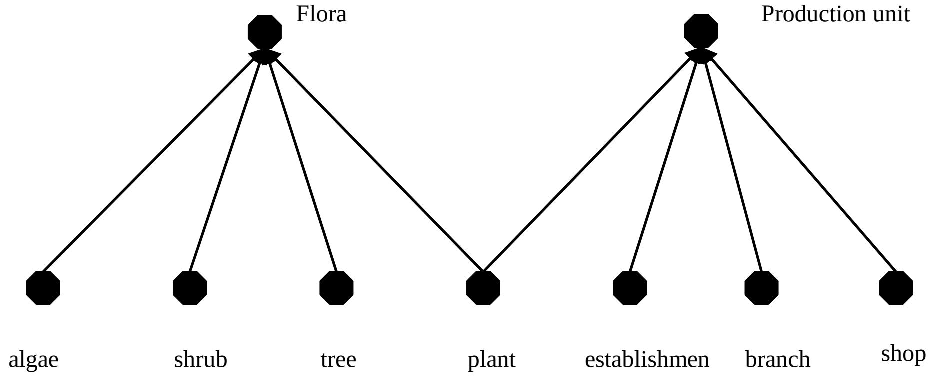
**Table 1. Semantic relationships among words important for coding.**

| **Semantic relationship** | **Explication** | **Examples** |
|---|---|---|
| Hyperonymy / hyponymy | A is generalization of B (A is a hyperonym of B; B is a hyponym of A) | Cars, trucks, ships are examples of vehicles |
| Synonymy | A denotes the same as B | A spud is a potato |

Synonymy expresses semantic equivalence. It should be stressed that synonyms in this paper are defined in a more general way than usual. They do not distinguish between verbs and nouns; nor between different tenses of verbs, singular and plural forms, cases of nouns (nominative, genitive, dative, etc.)

The C-words and D-words belong to different worlds, in a sense. The C-words are used to describe codes (via concepts) and the D-words belong to the world of descriptions (possibly after having been spell-checked and -corrected; see Section 6). To connect the two ‘worlds’, use is made of semantic relationships, i.c. synonymy and hyperonymy. There are also homonyms. They belong to two or more ‘camps’, that is, groups of synonyms. See Figures 6 and 7.

**Figure 6. Synonyms and homonyms**



The following example about a homonym that is relevant in a coding context concerning business activities.

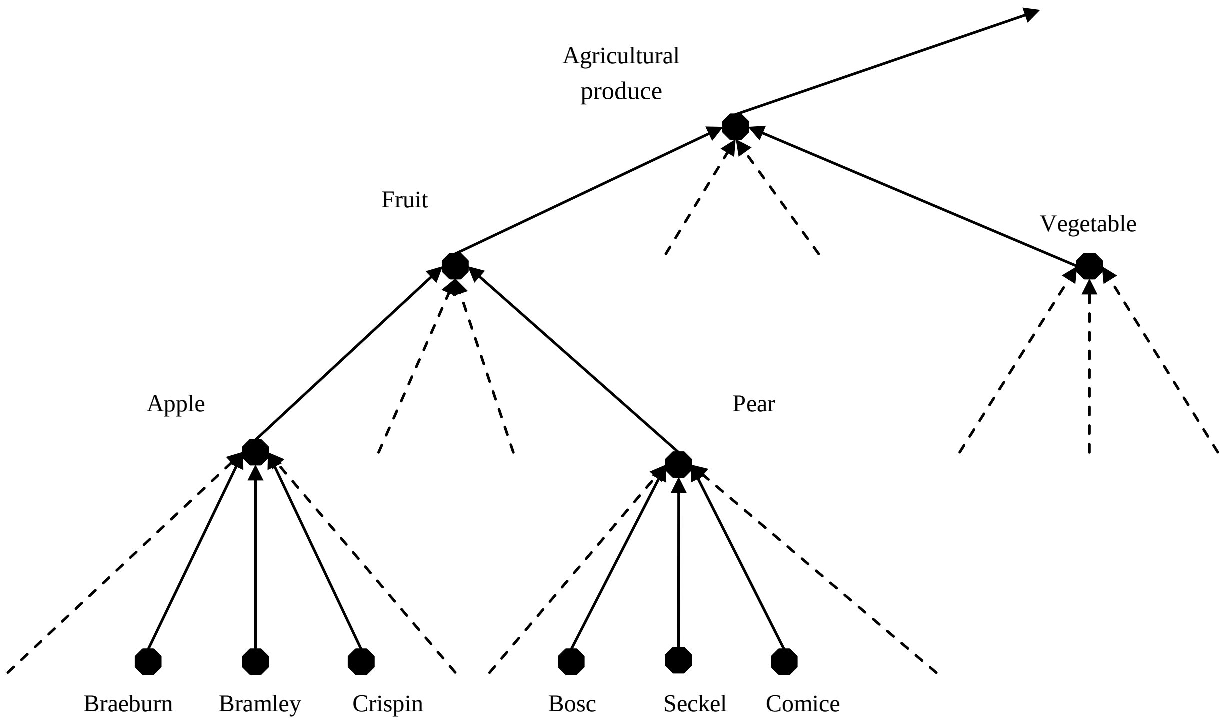
**Example 4.1.** The word ‘plant’ has several different meanings. First of all it can mean a ‘factory’, ‘mill’, etc. Or it can refer to a biological entity, which has its own set of synonyms (together with ‘flower’, ‘tree’, etc.). ■

We assume that the set of C-words form a subset of the set of D-words for a particular coding problem. This is simply realized by letting each C-word also a D-word. Of course, this makes sense only in case these words belong to the same language, although it would not harm if they do not.. Usually the set of C-words will be much smaller than the corresponding set of D-words. In practice, not all D-words will be connected to C-words. Some D-words contain no or little specific information, and are of no use (directly or indirectly) by defining C-words, and in the coding process. These words are called stop words. We will keep them in the collection of D-words as it is useful to know the stop words in a particular application (see Section 6).

Hyperonymy is useful in coding, as it is an embodiment of generalization. In case a respondent gives an answer that is too detailed for the coding purposes, it allows deducing an answer at the target level for the coding process. One should be prepared for the situation that a respondent might give an answer that is more detailed than the level required, unaware of the classification or the level of detail he should answer. But it is somewhat silly to ask again for an answer if the required result can be deduced by the coding system.

Suppose that a respondent in a survey about business activities answers that he ‘grows apples and pears’, but that this is too detailed for the purpose of the coding and that the required level is ‘growing fruit’. The answer given implies that, and we would like the coding system to deduce this from the answer provided. Given that the semantic network contains ‘fruit’ as a hyperonym for ‘apples’, ‘pears’, ‘prunes’, etc., this would be possible. Even a more detailed level should be contemplated to be part of the semantic network. Another respondent may reply that he ‘grows Braeburns and Reinette d’Orléans’. Ideally the coding system should recognize ‘Braeburn’ and Reinette d’Orléans’ as cultivars of apples. A cultivar, in this particular case, is a hyponym of ‘apple’.

**Figure 7. Agricultural products as part of a goods classification**



To illustrate a different kind of relationship that exists among words, consider the following example about synonyms.

**Example 4.2.**The words ‘produce’, ‘make’, ‘fabricate’, ‘manufacture’ and ‘create’ as well as related words, could be considered synonyms. Some of these ‘related words’ are given below.

- *Related to ‘produce’*: ‘produced’, ‘producing’ , ‘producer’, ‘producers’, ‘production’, ‘product’, ‘products’;
- *Related to ‘make’: ‘*made’, ‘making’, ‘maker’, ‘makers’;
- *Related to ‘fabricate’*: ‘fabricated’, ‘fabricating’, ‘fabricator’, ‘fabricators’, ‘fabrication’, fabrications’;
- *Related to manufacture*: ‘manufactured’, ‘manufacturing’, ‘manufacturer’, ‘manufacturers’, ‘manufactory’, ‘manufactories’;
- *Related to create:* ‘created’, ‘creating’, ‘creator’, ‘creators’, ‘creation’, ‘creations’, ‘creative’, ‘creativity’. ■

All these words can be considered members of a set of synonyms. Words like ‘factory’ and ‘plant’ (in a specific meaning) can also be defined as members of the same set of synonyms.

In case of synonymy we are dealing with a relationship that is symmetric. In case of an asymmetric semantic relationship (like hyponymy/ hyperonymy), we represent it in such a way that the direction is from the more specific to the more general. In that case, if a word is connected to another word, and if the specific word is triggered, the hyperonym will be triggered as well.

It is possible that a group of words have a common hyperonym but this hyperonym is not a C-word. In that case one can invent a D-word to represent this set. This hyperonym should then be used as a hyponym in another relation, linking it ultimately to a C-word. Otherwise there is no point in storing that such relationships exist (i.e. finding hyperonyms for stop words is a superfluous activity).

**Remark 4.1** There are more semantic relationships than Table 1 indicates. For instance there are antonyms (opposites), meronyms (indicating that something is a part of something bigger) and holonyms (the dual of meronyms). However, such relations are not required for the approach described in the present paper. But they may have potential for an extension of this approach. Antonyms may be of use in dealing with exceptions (see Section 5.1). Meronyms / holonyms may be used as an alternative way to generalize notions, apart from the hyponym / hyperonym relationship that is used in the present paper. If a company produces ‘spark plugs for cars’ than one cannot conclude that this company ‘manufactures cars’ on the basis of the fact that a spark plug is part of a car. One only can conclude that it produces a car part. But then one uses the fact that a ‘spark plug for a car’ is a ‘car part’, using hyperonymy. ■

For C-words we can ask what relations can or may exist between them. First of all it can be avoided that they are synonyms, because the sets of synonyms could then be combined. Also it can be avoided that C-words are homonyms, by choosing them all differently. C-words can be hyperonyms or hyponyms. In fact this relationship and its inverse are important to change level of detail

## 4.2 Derived relations for concepts and codes

In this section we apply the hyperonym / hyponym relationship at the C-word level to create relations at the concepts level.

Suppose that $\{A_1,..., A_n\}$ is a concept, where the $A_i$ are C-words for $i = 1,..., n$. Further we let $\leq$ denote a binary relation on the set of C-words, such that $a \leq b$ means that $b$ is a hyperonym of $a$, or equivalently that $a$ is a hyponym of $b$. Then if $A_i \leq a_i$, for some $i$, then we have

$$\{A_1,..., A_i,..., A_n\} \leq \{A_1,..., a_i,..., A_n\}. \tag{4.1}$$

If also $A_j \leq a_j$ for some $j \neq i$, we have

$$\{A_1,..., A_i,..., A_j,..., A_n\} \leq \{A_1,..., A_i,...a_j,..., A_n\} \leq \{A_1,..., a_i,..., a_j,...A_n\}, \tag{4.2}$$

and also

$$\{A_1,..., A_i,..., A_j,..., A_n\} \le \{A_1,..., a_i,..., A_j,..., A_n\} \le \{A_1,..., a_i,..., a_j,... A_n\}\ . \tag{4.3}$$

See Figure 8, which shows a Hasse(-like) diagram for this situation.

We also have that deletion of a C-word from a concept, yields a more general concept. So we have for instance

$$\{A_1,..., A_i,..., A_n\} \le \{A_1,..., \breve{A}_i,..., A_n\}\ , \tag{4.4}$$

where $\breve{A}_i$ denotes the removal of $A_i$ .This latter kind of generalization can be seen as an extreme form of the one introduced in (4.1). The generalizations can be applied repeatedly to build a chain. Applying transitivity yields single step generalizations consisting of several smaller step generalizations. In this way the set of concepts and their generalizations form a partially ordered set.

It is clear that in this way for a given concept many generalizations can be found. But a lot of these are not very useful, as they do not correspond to any code. In order to get the useful ones, we have to look at the codes in the classification and the way in which they are defined in terms of concepts.

As we have seen before, the codes in a classification are usually associated with nodes in a (directed) tree structure. With the codes we also have associated concepts that define them, and the concepts are represented in terms of C-words. See Figure 9.

**Figure 8. Generalization of concepts**

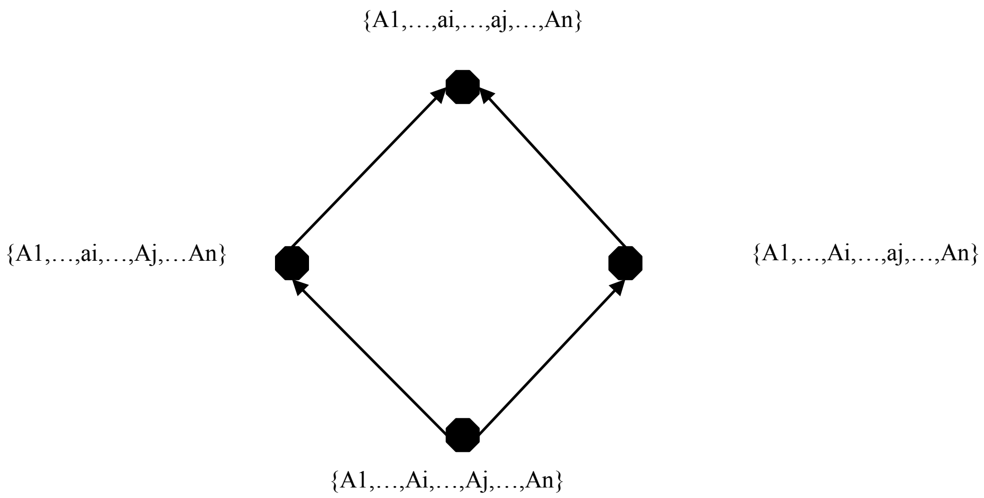
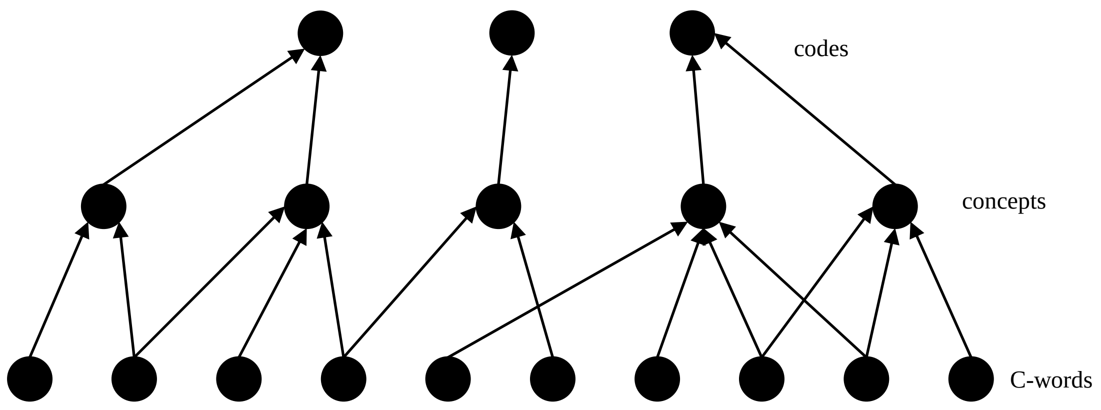


**Figure 9. Hierarchy of codes concepts and C-words.**



The next task is to use the (directed tree) structure of the classification to link the concepts associated with the nodes that are directly linked in the tree structure using the semantic relationships hyperonymy/hyponymy. See Figure 10 for an example.

In Figures 10 and 11 we have hyperonym / hyponym relationships between certain C-words, where:

$$A \leq a, B \leq b, C \leq c, D \leq d \ . \tag{4.5}$$

So in this case, $a$ is less detailed than $A$ , or put in another way, $a$ is a hyperonym (superordinate) of $A$ . Similarly, for the other pairs of C-words appearing in (4.5).

Once concepts associated with the various nodes in the classification have been linked together in this manner, we have finished a semantic network for this

**Figure 10. Concepts associated with parent-child nodes.**

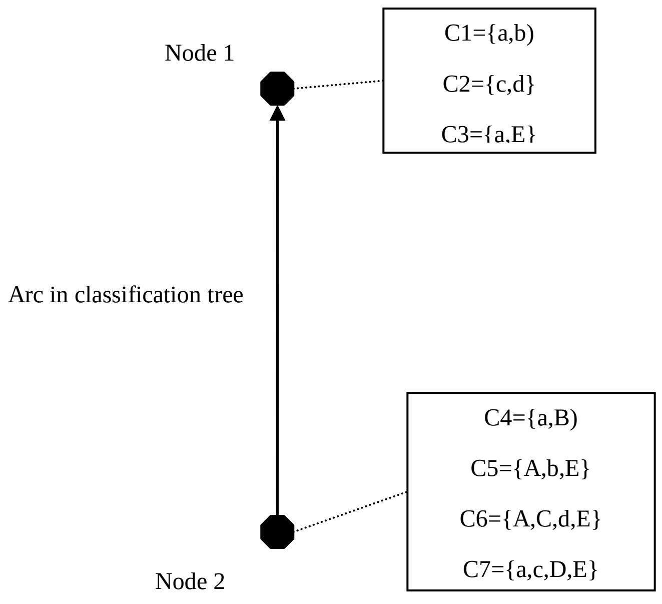
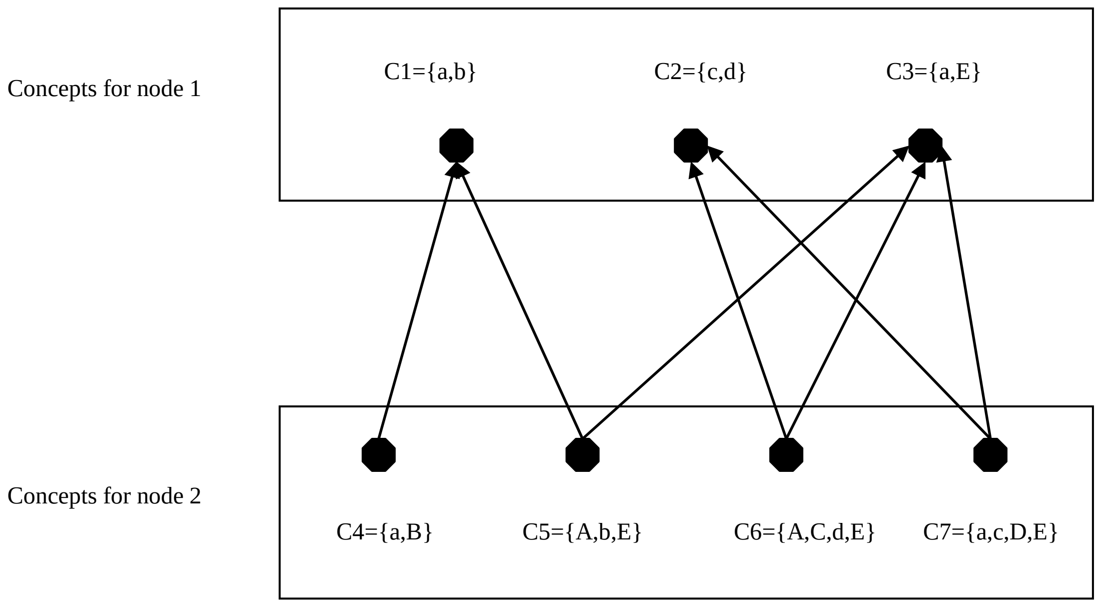


**Figure 11. Semantically linked concepts associated with parent-child nodes.**



classification. In practice, such a network is likely to be in a state of permanent adaptation. New descriptions of codes in terms of concepts can be defined, and these have to be tied into the existing semantic network. Also new D-words will be found and have to be added to the network. It will also be a never ending task to check the correctness of the semantic network, initially and after any change that has been made to its structure or its content.

## 4.3 Constraints

This section is to make the reader aware that certain constraints exist for a semantic network for coding in order to be consistent. It should be admitted that the intention of this chapter is rather modest. It tries to call for attention to a specific problem, but it does not solve it at all. So instead of discussing a complete set of such constraints, we, more modestly, can only offer some examples of constraints to hold. A thorough discussion of this issue is better left to a special paper that is entirely devoted to this topic.

So we only contend ourselves with providing some examples of the constraints we have in mind for semantic networks for coding.

A first example concerns the definition of codes. First look at the set of concepts $V_{con}$ and the set of codes $V_{cod}$ for a given classification. As all concepts should be useful, the first requirement is that each concept should be associated with at least one code. But this is not enough. In order to be unambiguous, we must require in addition that each concept is associated with at most one code. So in fact we have that each concept is associated with exactly one code. Furthermore each code should be described by at least one concept. All this can be more formally expressed by stating that there exists a surjective function $\kappa : V_{con} \rightarrow V_{cod}$ . This implies that the size of $V_{con}$ is not smaller than that of $V_{cod}$ .

Furthermore the semantic relations on the set of concepts should comply with the hierarchic structure of the classification to which the codes comply. To illustrate, consider Figure 12.

It shows two codes, code 1 and code 2, and two concepts concept 1 and concept 2, characterizing code 1 and code 2, respectively. There are three maps (functions) $\kappa, \lambda$ and $\sigma$ , where $\kappa$ is as defined above. The map $\lambda : V_{cod} \to V_{cod}$ is defined by the classification, for each $c \in V_{cod} \setminus \{root\}$ , $\lambda(c)$ is the code associated with the (unique) parent node. We assume that $\kappa$ and $\lambda$ are given. In case a semantic relation $\sigma$ (as discussed in the previous section) has been defined linking concepts 1 and 2, we require that the diagram in Figure 13 is commutative. This means that

$\kappa \circ \sigma = \lambda \circ \kappa$ . In case no semantic relation has been defined, it is advisable to find out if this is still possible. It may require finding concepts defining code1 and concepts defining code2 that can be linked semantically. Or it could require that a new concept1 for code1 or a new concept2 for code2 needs to be added, so that a semantic link can be established. In case such a link $\sigma$ is found, it should satisfy the commutativity property shown in Figure 12.

The advantage of such semantic links is that they can replace links derived from $\lambda$ which are in a sense extraneous. The semantic links are internal and they allow inferences and generalizations ('spreading action') within the semantic network. In fact we ideally should have that all concepts associated with code1 are linked semantically with those of code2. In case a concept $K$ associated with code 2 is not

**Figure 12. A commuting diagram for codes and associated concepts**

$$\begin{tikzcd}
\text{Code 1} & \text{Concept 1} \arrow[l, "\kappa"'] \\
\text{Code 2} \arrow[u, "\lambda"] & \text{Concept 2} \arrow[l, "\kappa"] \arrow[u, "\sigma"']
\end{tikzcd}$$

linked semantically to a concept associated with code1, we could use $\lambda$ to generalize from $K$ but this does generally not yield a single concept, but a set of concepts, namely all those associated with code1. So $K$ may be linked in this way to a concept associated with code 1 to which it is not semantically related. This is to be avoided, if not to be forbidden.

In fact, there is even a stronger requirement concerning the concepts associated with the same code than that they should be different. This requirement asserts that no (nontrivial) semantic relations should exist between concepts associated with the same code. .

Another kind of constraint involves concepts and the C-words from which they are constructed. It is required that there are no two concepts consisting of the same C-words. If we put this in another way, we require that, if $V_{cwor}$ denotes the set of C-words and $\wp V_{cwor}$ denotes its power set, there is a map $\vartheta : V_{con} \to \wp V_{cwor}$ that is injective.

Another constraint that we mentioned before concerns the choice of C-words. They must be chosen in such a way that they are never homonymous. In this respect D-words and C-words are different. D-words may have homonyms, but C-words never have. If we leave out the homonyms from the D-words (that each link to two or more D-words) as well as the stop words (that link to none of the D-words) the remaining set of D-words is such that each of them link to exactly one C-word, by definition.

## 5. Some examples

It is possible that hyponyms or hyperonyms of a D-word are also D-words themselves. This is necessary when these D-words appear in concepts associated with codes at different levels of detail. An example of this situation occurs for instance in the coding of business activities.

**Example 5.1** Consider the following activities related to vegetables:

1. ‘growing of vegetables’ corresponds to a unique code A , no matter what the vegetables being grown are, potatoes, lettuce, beet roots, onions etc. (The reason is that the growing of vegetables is very similar, no matter the variety that is grown. For the application intended at least, these growing activities are considered the same. This is a choice.).
2. ‘processing of potatoes’ leads to a unique code B, which is different from ‘processing of grain’ or ‘processing of beet roots’.(The reason is that these processes are quite different, and important to make for the application intended).
3. ‘wholesale of seed potatoes’ and ‘wholesale of edible potatoes’ correspond to different codes C and D. (The reason is that they operate on different markets, and for the application intended, this distinction is important, apparently). So ‘wholesale of potatoes’ would be ambiguous, in the sense that no unique code would correspond to this description. What is lacking is

the type of potatoes concerned, i.e. seed potatoes or edible potatoes. The distinction between these forms of wholesale is made since they are directed at different markets, roughly farmers and retail.

In summary we can define concepts involved in the codes A, B, C and D, as follows:

Code A = {{vegetables, growing}},

Code B = {{potatoes, processing}},

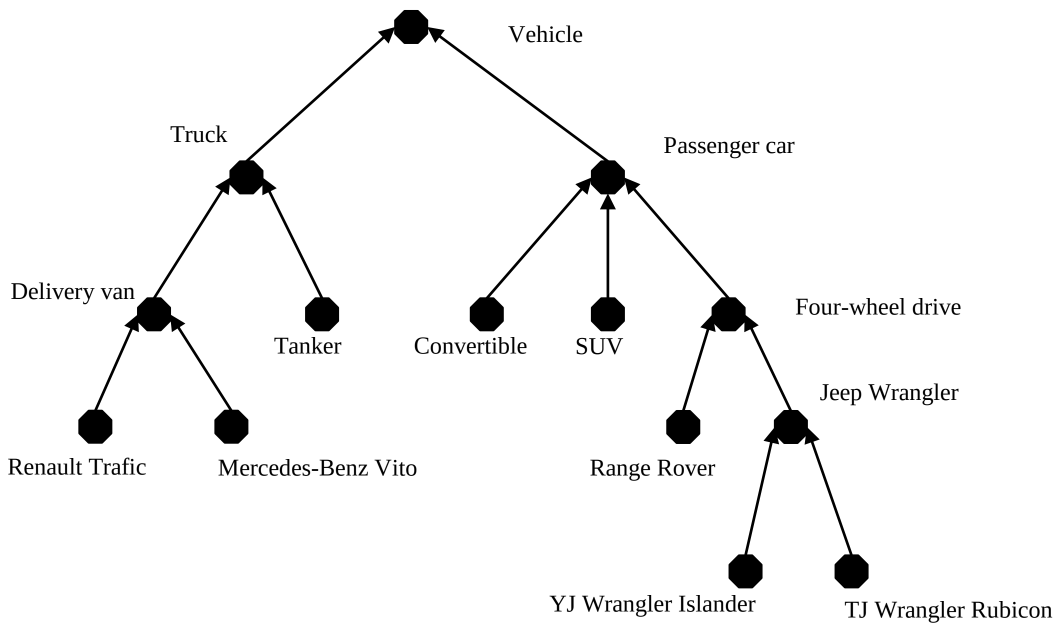
Code C = {{edible, potatoes, wholesale}},

Code D = {{seed, potatoes, wholesale}}.

So we see that in the first example it is not necessary to specify what vegetable it is that is being grown. In the second example to mention that potatoes are processed, is just the right level of detail and a single code corresponds to this activity. In case of wholesale, it needs to be specified what kind of potatoes this activity concerns: seed potatoes or edible ones. It is possible (and useful) to define the concepts {edible, potatoes} and {seed, potatoes}.Both are hyponyms of potatoes. ■

**Example 5.2** Careful attention should be devoted to the organization of D-words. The first idea is perhaps to simply organize them in disjoint sets, each of which is a set of synonyms. We could, for instance, have a group of words related to 'vehicle', to which such words as 'trucks' and 'passenger cars' (among several others). The 'trucks' can be divided into 'delivery vans' and 'tankers' (among several others). The 'passenger cars' consist of 'convertibles', 'SUV's and 'Four-wheel drives' (among others types). Several brands have 'Delivery vans', such as Renault (the 'Trafic') and Mercedes-Benz (the 'Vito'). Among the four-wheel drives we have the 'Range Rover' and 'Jeep Wrangler'. This latter one consists of various types, among them the 'YJ Wrangler Islander' and the 'TJ Wrangler Rubicon'. If there is no need to have such an elaborate tree structure for vehicles, one could use a flat directed tree, with 'vehicle' as root, and all the categories mentioned as synonyms of this word. As representative word for the set of synonyms 'vehicle' could be used (suppose that this is a C-word). However, this supposes that in any combination with, say, activities such as 'manufacturing', 'sales', 'repair', etc, vehicles would have to be specified at the same level of detail. However, a tree structure as described and as depicted in Figure 13, would be easier to maintain. ■

**Figure 13. A vehicle tree.**



The following example is about clothes. This example illustrates a few interesting aspects, relevant for automatic coding in general.

**Example 5.3** Clothing is a collective term that comprises many different things. See Figure 15. The distinctions that are made depend on different criteria. The use of the cloth can be important (industrial or professional use – such as uniforms, or civilian clothes), or the gender (male, female) of the persons they are intended for, or their age groups (grown-ups, children, babies), how the clothing is worn (garments, underclothing). The example in Figure 13 is in fact restricted in fact to human clothing. There is also animal clothing, e.g. for horses, for dogs, cats, etc. This would be part of the 'other cloths' category, among other types of clothing not mentioned here such as 'sports clothing', 'leisure wear', etc.

These distinctions are important in view of the activities. For instance, in case of manufacturing, it is important to distinguish between 'garments' and 'underclothing', whereas for retail the distinction between 'men's clothes', 'women's clothes' and 'children's clothes' is relevant. This is comparable to the example concerning the business activities involving agricultural products.

Another point to note is that the various types of clothes mentioned in Figure 14 are not disjoint categories. For instance, babies are children. Children are boys or girls and hence male of female, etc.

The 'other clothes' category is an example of an exception category. In this case it is very useful, as it seems to be well neigh impossible to list all types of clothes that exist. Besides for the coding problem at hand it is possibly only of interest to make some distinction in clothing and lump the rest into a single remainder category and treat that in a separate way.

**Figure 14. Clothes example**

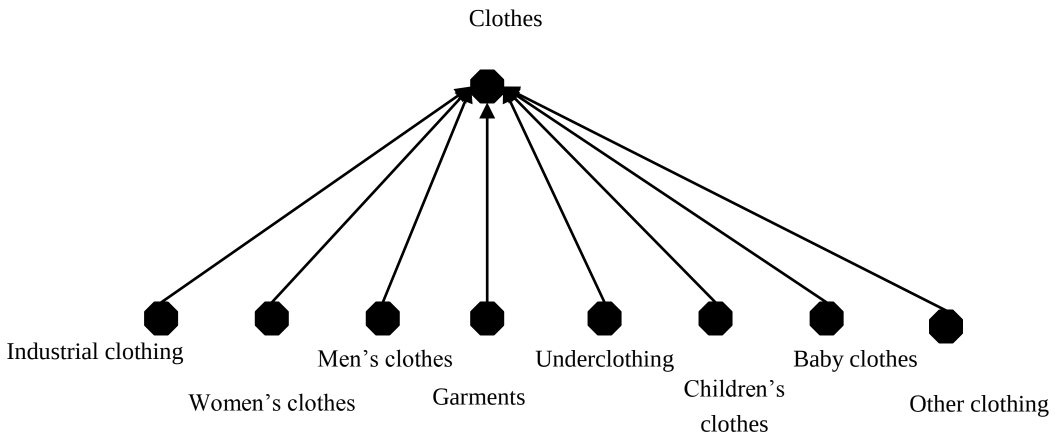
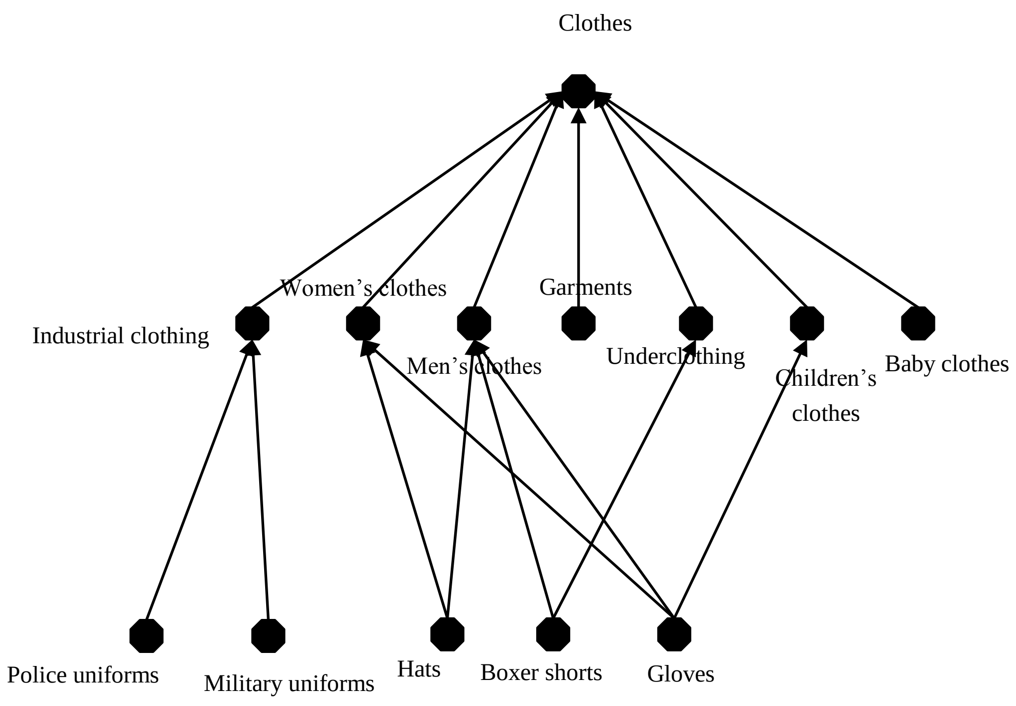


Figure 14 is presented as a very flat directed tree. But with some additional clothing types distinguished a more general structure is required. Because the nodes are not disjoint this is not a tree, but a directed acyclic graph (DAG). Figure 15 shows a (partial) elaboration of the clothes example in Figure 14. It elaborates some of the categories shown in this figure. This elaboration is far from complete. But it shows that the structure obtained is not a directed tree, but a DAG, as certain objects, such as 'hats', belong to more than one category, in this case 'women's clothes' and 'men's clothes'.■

**Example 5.4**. It seems to be a good idea that the most detailed codes used in the classification at hand are atomic, that is, should pertain to a single business activity, product, service, etc. In practice this is, however, not always the case. One may, for instance, find a code in a classification for 'selling and repairing cars', because these combined activities occur in practice and are equally important for some businesses, in addition to the activities 'selling cars' (without repairing them) and 'repairing cars' (without selling them). If the main activity would have to be chosen, neither 'repairing cars' nor 'selling cars' would give an adequate picture of the activities of some businesses. But it must be admitted that the concept of 'atomic activity', 'atomic type of education' is not as clear-cut as it may seem at first sight. It also depends very much on the application. It is similar to the problem of distinguishing between atomic and composite entities. Here, as well, what is an atomic entity depends on the application

So it is to be assumed that there will always be problems with this aspect of descriptions. ■

**Figure 15. Partially elaborated structure for clothes (a DAG)**



**Example 5.5.** It should be noted that sometimes descriptions collected for other purposes than statistical ones, there may be a bias, or the description may be deliberately more general. This is a more serious problem that can be done little about in itself (except for asking for information from an alternative source, that is more neutral and provides more honest and realistic answers). ■

## 6. Coding with a semantic network

In this section we give a brief description of how automatic coding system could work. The description is mainly to give the reader an idea of the various steps that are needed. The description focuses on the happy flow, which is the best case possible. In practice many exceptions can occur and a real system should deal with these. Some of these exceptional cases are discussed as well, to give the reader an idea of what to expect. The discussion is not complete, however, neither concerning the range of exceptions to deal with, nor with the completeness of the discussion for the exceptions considered.

In Section 6.1 we describe the so-called happy-flow, which deals with the situation when a description contains the information to find exactly one code (which then automatically is assumed to be the correct one). In fact, we consider a simplified situation here, namely when there are only positive descriptions, that is, without exceptions. This is not because we think exceptions (and exclusions) are not important. It is because they concern a rather specific technical topic. For that reason this subject is not treated in the main text, but in Appendix B.

## 6.1 Happy flow

The automatic coding process using a semantic network in case a description yields a single code in the classification used ('happy flow') is roughly as follows:

1. Start with a description S, a string of characters.
2. (optional) Put S through a spell-checker. Resulting string: S*.
3. Segment S* into tokens /words.
4. Check that each of these tokens / words in the string is a D-word.
5. Find the C-words that these D-words imply, using the semantic network.
6. Find the concepts that these C-words imply, using the semantic network.
7. Find the code K in the classification used that these concepts imply, using the semantic network. In case several codes are triggered, the most detailed one should be chosen
8. Attach the code K to the string S.

## 6.2 Analysis of the process flow

This description in the previous section contains the 'happy flow' where each step is ideal. The input string S apparently contains the right information to lead to a single code. Although it is not guaranteed that this code is correct, it is usually accepted as the code that corresponds to S.

In practice, however, one usually finds exceptions to the ideal steps. A major reason for this is because respondents do not know (and are not expected to know) the underlying classification and consequently may be unaware as to what information to put into their descriptions and at what level of detail. Another, more general, reason is related to the language respondents use, which may contain jargon, slang or words from dialects or even other languages. Or they may have spelling errors in their answers. Respondents may also use exceptions in their answers, which add another level of complications of coding process. An exception is used as a means of getting an easier description. For instance, one may say 'processing of fruit, except grapes and berries' instead of providing a list of fruits that are actually included in the processing, that is much longer. Section 5.1 goes into this topic a little deeper. See also Hacking &Willenborg (2009).

We discuss each of the steps in the happy flow briefly and point at certain complications that might arise.

**Step 1**. Ideally the string should contain numbers, alphanumeric characters, interpunction (dots, comma's, hyphens, etc). But maybe it contains characters outside this range as well. These should be handled as well, in order to avoid problems later on in the coding process. This step could be viewed as an ETL step, or a data cleansing step.

**Step 2**.  This step is not exclusive for coding. It is a problem in the area of spell-checking and correcting. Tools exist that handle this problem

**Step 3.**  The segmentation process is called tokenization (Mitkov, 2004, ch. 10). The aim is to recognize tokens, in our case words in the descriptions. If the descriptions are longer pieces of text, it becomes important to consider interpunction as well. Also the grammatical structure of descriptions could then be taken into account. In the present paper we assume that this is not necessary, as the descriptions are assumed to be short.

This tokenization problem transcends automatic coding. We shall not go into it any further here. We can also assume this problem to be solved, in the sense that adequate software exists for this step.

All sorts of errors are possible due to the incorrect spelling of words, the incorrect usage of interpunction, the incorrect splitting words, the incorrect concatenation of words, etc. A problem in some languages, like Dutch or German, is that long words may appear in descriptions (if correct spelled). The problem is that these long words may pose spelling problems for many people. Under the influence of English they tend to break up long words into smaller components. The opposite also happens. If one does not pay special attention to this particular problem (in Dutch, say) the automatic coding system is bound to under-perform. To cope with these spelling problems one may have to introduce an extra spell-checking and spell-correcting step. Appendix C goes into this area a bit more.

**Step 4.**  This step checks if all tokens / words found in the previous step can be linked to D-words currently in the. There are two possibilities for each of these tokens.

1. All tokens / words can be recognized, and so the entire description is recognized:
2. At least one of the tokens / words is not recognized as a D-word. Those  that are not, should be set aside and an expert has to add them to the semantic network. This is an opportunity where the semantic network can 'learn' from new input. The description should be put on a special list, and passed again to the coding system after it has been updated with the new tokens (now D-words).

In case of a happy flow, all tokens in the description are recognized as D-words. It is important to check that this is the case. A token not recognized as a D-word cannot be discarded, as it may very well be a new D-word. Such a word has to be incorporated into the semantic network, which includes its linking to the C-words in there (and hence to concepts and codes). In case not all D-words in a string are used, a situation arises that may lead to coding errors.

So tokens in the description that are not recognized should be dealt with first, that is, before the description is used for coding. This requires experts to look at these tokens / words and 'feed' them to the semantic network.. This means that new words have to be added to the set of D-words and that these new D-words have to be linked (through semantic relationships) to C-words in the semantic network. It is possible, of course, that some of the unrecognized tokens turn out to be stop words, so are actually not used in finding codes.

After the semantic network has been updated the description needs to be parsed again. Now all tokens should be recognized as D-words.

**Step 5.** If the entire description is recognized, the D-words it consists of are linked to C-words (or: activate C-words), unless they are stop words. In the latter case they are simply discarded. Note that these words should also be recognized first as D-words, to make sure that all tokens / words in a description are used..

**Step 6.** From the activated C-words all concepts they activate should be found, which should be done automatically by the semantic network.

Problems encountered in this step are at the heart of the semantic network used. It is very well possible that some semantic relationships are lacking and that others are wrong. It may also be the case that the semantic relationships that have been defined are too limited and should be expanded. The description may not be ideal: it may lead to more than one concept (belonging to different codes; if they all belong to a single code then there is no problem) and hence to more than one code; or it may correspond to no code at all.

**Step 7.** There are several possibilities that can occur when trying to find a code triggered by the description.

1. No code is triggered. This can occur for several reasons. It starts with the observation that there was no code for which a combination of defining concepts was triggered. This indicates that some vital C-words in some concepts are missing. This requires additional information from the respondent (if available) to arrive at a code
2. The description triggers exactly one code .at the required level of detail. This is the ideal case, described in the happy flow. (It is actually a bit more complicated; see the comment below.)
3. The description triggers more than one code at the required level. Apparently the description is not specific enough. In this case additional information is also required to arrive at a single code.

The problems with this step may be that either no code is found or more than one. In the latter case crucial information is possibly lacking from the description. Without any additional information (provided by the respondent, or after inspection of a coding expert) no single code can be found and the coding is inconclusive. Another possibility is that no code is found to match the description. This can happen if no combination of concepts is found that correspond to a code. This in turn can have various causes, ranging from the case that not a single C-word is found to correspond to the description, to the case that several concepts are found to match with the description, but together they do not define a code, nor does any sub-combination of the concepts. It is even possible that a single code is found, but on closer inspection this turns to be wrong. In a sense this is the worst case, as a lot of things may be wrong then. We will not delve into this further, as it takes is too far from our goal of this paper. These topics need to be addressed separately, and in all necessary detail.

**Step 8**. In case no (unique) code is found at this step, it should be decided what to do: to ask for feedback information from the respondents (if available) or have coding experts look into the problem. They may be able to detect vital information that may lead to another code; or cast doubt on the code that has been found in the first step. ■

The comments above should also give some indication about the exceptions that may occur in the various steps. Handling them is a major task, but not for the present paper to describe.

## 7. Summary and discussion

The idea behind using a semantic network in automatic coding is to separate the data (which is the semantic network in this case) from the software for navigating it. The navigation program is supposed to be independent of the particular contents of a semantic network. The semantic network itself is application (that is, classification) dependent. It only makes sense to build a semantic network for a particular application (classification) if this is to be applied on a recurring basis, as building a semantic network for coding requires quite some effort. The navigating software should be available in order to use it. This may require extra effort to build this software, and maintain it as long the semantic networks are being used for automatic coding.

It should be stressed that automatic coding is not the only type of coding that used in practice. This type of coding is intended to handle the bulk of the coding work for a typical application. It requires that the input is in the form of digital descriptions, i.e. strings of letters that are machine readable. In case they are on paper, they should first be digitized. This is basically transforming the input format. It is not the same as interpreting what is actually written in terms of the classification used; that would be manual coding. Manual coding used to be the only way of coding in ‘the old days’. But it cannot be completely discarded at present; in fact, it should be used where the automatic coding fails to produce an answer.

The advantages of automatic coding over manual coding are several (if it works!): more material can be coded in the same time; it can be done 24 hours a day, 7 days a week; the decisions to derive a particular code (or the failure to do so) on the basis of a description, can be documented, and can be analysed afterwards. Of course, this possibility should be facilitated by the coding software. It seems advisable to build in the possibility to keep such audit trail information. Not only does it make the coding process transparent, it also allows the maintainers of the coding system (both the semantic network and the navigating software) to check the quality of the coding results, look for possible errors, and produce improvements.

Typical applications for automatic coding are ‘occupation’, ‘education’, ‘business activity’, ‘leisure activity’, ‘type of disease’, ‘cause of death’, etc. In all cases we are dealing with a variable that has a quite large domain, that is, set of possible answers.

The idea is to use a vocabulary of C-words to form concepts. Each concept is a set of C-words. In turn, each node in the classification tree is supposed to be represented by one or more concepts. This defines the ‘target side’ of the coding problem.

At the ‘source side’ we deal with descriptions in some natural language, which we assume are strings of characters, that is, letters from an alphabet. The problem is to extract as much information from such a description as possible. In doing so one faces language problems due to imperfections in the descriptions (ranging from spelling errors, through the use of words that are not recognized by the system), but also problems due to the fact that the respondent is unfamiliar with the classification being used to code the descriptions. This may manifest itself through over-complete descriptions, that is, one that contains more information than is needed for a single code, or under-complete, that is, contains insufficient information to lead to a (unique) code.

Some of the language problems faced can be tackled by employing a spell checker and corrector. In case the semantic network does not contain certain important words it should be extended. There is always a possibility that one runs into words or expressions that are new for the semantic network. It is a part of its learning capability. This effect of encountering new words or expressions for a particular application will be less likely the longer the coding system is used. But one should always be prepared for it.

The second problem (ignorance of the classification) can only be remedied by interaction with the respondent, to solve ambiguities and request additional information. But interaction with the respondents is not always an option. In a typical situation in which automatic coding is applied, the data have been collected and are being processed as they are delivered to the statistical institute to process them, and in case a problem arises it is not always possible to contact the respondent involved again and ask for additional information. In that case human coders try to make sense of the information available, in case the automatic coding program fails to do so.

It should be noted that the ‘source side’ of the coding problem is (natural) language dependent, in the sense that the choice of words and phrases is not limited, whereas the ‘target side’ is not, in the sense that it works with a limited vocabulary. Of course, the C-words used at the ‘target side’ are chosen from a language, but at the ‘input side’ one may have to deal with different languages in which the descriptions are given, e.g. in Dutch / Flemish, French and German in Belgium, in various languages in the territory of the European Union, in English, French and Inuit in Canada, etc. One can then choose one language from which to take the C-words to describe the concepts, and the concepts in turn to describe the codes acting in the classification that is being used. For each of the languages mentioned one has to arrange that descriptions in that language can be handled, i.e. translated, into the target language.

Important practical issues with the approach discussed in this paper are the following:

- How to develop a semantic network for a particular application?
- How to maintain it?
- How to check that the coding system is trustworthy, that is, free of errors leading to wrong codes?

This is both a matter of the semantic network and of the navigating software, as both may contain errors. These are important issues that require attention in separate papers. As far as the trustworthiness of the coding is concerned, one should be aware that there are two situations to consider, comparable to type 1 and type 2 errors in testing (see for instance Mood, Graybill & Boes, 1963, p. 405). The first kind of error is easier to note: a description that does not actually yield a code, but that, on closer inspection, should. At least then it is clear that no code is found. In case a code is found, but this is not correct on closer inspection, one has a bigger problem, as this error may go unnoticed. This implies that the attention should not exclusively be focused on the descriptions that do not lead to a code, but also on those that actually have yielded a code.[2]

As a generalization of the current approach it is a possibility to take the order of D-words as they appear in description into account as well some syntactical information from the descriptions, such as interpunction, so that only certain combinations of D-words are considered. In the approach described in this paper only the appearance of D-words is important, not their location in the description. It is possible that this results in wrong codes in certain situations. It is possible to cope with this situation when codes can be characterized not as a set of concepts, but as a set of combinations of concepts. For then it is possible to distinguish $(x, y)$ from $(y, x)$, as follows: $(x, y) = \{x, \{y\}\}$ and hence $(y, x) = \{y, \{x\}\}$, which are different objects in case $x \neq y$.[3]

The aim of coding is to relate the descriptions provided by respondents to specific codes in the classification used. But it is only possible to relate a description to precisely one code in special cases. This is particularly special if one realizes that the respondents are supposed to be ignorant about the classification being used. So it is

[2] This is comparable to (and in a sense, an example of) the case of response and nonresponse in surveys. It is easy to focus on the nonresponse, as this is directly visible. But the response should not be taken for granted. Its quality is not by definition beyond dispute. Data editing is the methodology that checks the plausibility of the responses recorded. In practice this usually amounts to 'internal consistency', not 'agreement with reality', i.e truth.

[3] This is a well-known 'trick' in set theory.

likely that respondents give the wrong information, insufficiently detailed information or information that has too much detail, or a mix of two or more of these. The case of too much detail can be handled already in the current approach by taking more hyponyms into account at levels beyond the required ones.

So without help, asking respondents to give descriptions that lead to unique codes in a coding system they are not supposed to know is like firing shots in the dark at an invisible target. Hitting this is a matter of luck rather than of marksmanship.

So a fully automatic coding system used to interpret descriptions that are not guided in some way, is bound to failure. There are several options to make improvements. After the initial description and the attempt to interpret it in terms of the classification used, it would be good to go back to the respondent if there is no code or more than one possible code that fits the description, and ask for clarification or additional information. However, this interaction is not always possible. The best one could do in such a situation is to let a human coding expert look at the descriptions that caused trouble, that is, that could not be related to a unique code.

Another option is to avoid free descriptions altogether, and use instead a limited number of simple questions with pre-coded answers. They should be able to cover the majority of cases.[4] These closed questions should be simple enough to be answered by respondents, still assumed to be ignorant about the classification used. The code can than be deduced. To check the coding results it would be advisable to ask the respondents to give a description in their own words, in addition to their answers to the closed questions. What a suitable set of closed questions would be (and whether they exist!) cannot be discussed in general. This depends strongly on the application area.

After these general remarks concerning automatic coding, we now turn to the coding method we proposed in this paper. The central problem of coding is to relate descriptions to codes, if possible, and with the help of automatic coding to do this linking fully or partially automatically. The first step is to dissect the descriptions and derive the words from it, and correct them if necessary. This step has little to do with coding proper. It is pure language processing, and in practice one should use all relevant information (and software) from this area. So for us the step to associate with a description a sequence of words (in the same order in which they appear in the description) is supposed to be solvable by existing linguistic technology.

So the problem we face is to try to find a code associated with such a sequence of words (from a description). Each code is represented by a set of concepts, where each concept is a set of reserved words called C-words. The first problem we may encounter is that a word is not in the vocabulary, i.e. the set of D-words. As the D-

[4] The remainder of the cases should be dealt with in other ways, for instance by manual coding. The questions should each have an ‘escape category’, which is open, i.e. not precoded. In this case a respondent always has a possibility to give an answer in his own words, in case a suitable one is not found among the precoded categories. To take recourse to such an escape category should not happen too often, though.

words are related to the C-words in a semantic network, unless they are redundant as they carry no or little information, it is likely that no unique code will be found that is associated with the description, respectively the sequence of words.

A problem encountered when implementing the approach described in the present paper is to setup a semantic network to link the descriptions to the codes in the classification that is used. One problem when the semantic network has been set up is to verify that it is correct. The semantic network will not be static after its initial setup. Rather it will be modified as a result of new input; this is its learning capability. Also as a dynamic entity, we have to be able to test that the semantic network is bona fide (or, with high probability) after each update. And we should be able to go back to the last stable or consistent state if necessary. This implies that the updates should be versioned and be stored as such. Also redundancies should be avoided or eliminated, although they are a nuisance rather than a problem. Useful references for gleaning through when building a semantic network are Sowa (1984, 2000).

Perhaps the current approach needs to be extended as follows. Not only should synonyms and hyperonyms / hyponyms of D-words be taken into account, but also of sets of concepts. This means that e.g. it should be possible to have concepts A, B, C, D, such that {A,B} is synonym of {C,D} or $\{A, B\} \leq \{C, D\}$ holds, without A synonym to C (or D) and B synonym to D (or C), etc for hyperonyms / hyponyms.

## Acknowledgements

This paper has a long gestation time, due to various circumstances. I am grateful to Hans van Hooff and Wim Hacking for discussions on automatic coding, at various occasions. To these colleagues and to Sander Scholtus I am grateful for their comments on earlier drafts of this paper. However, for any omissions or errors I am solely responsible.

## References

Hacking, W. & Willenborg, L., 2009, Contribution on coding to the Methods series, Statistics Netherlands, Heerlen and Voorburg.

Mitkov, R. (ed.), 2004, The Oxford handbook of computational linguistics, Oxford University Press.

Mood, A.M., Graybill, F.A. & Boes, D.C., 1963, Introduction to the theory of statistics, 3rd ed., McGraw-Hill-Kogakusha.

Sowa, J., 1984, Conceptual structures: Information processing in mind and machine, Addison-Wesley.

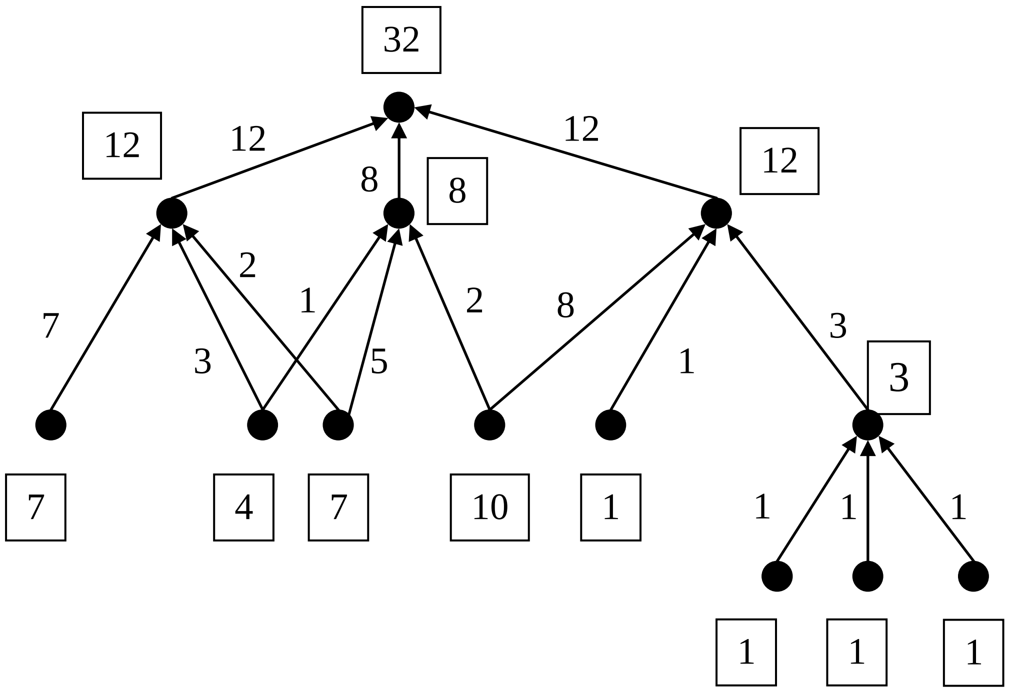
Sowa, J, 2000, Knowledge representation, Brooks/Cole.

## Appendix A. Directed acyclic graphs (DAGs)

A classification is usually a tree. (In our case it would be a directed tree.) But this is in fact not a necessity. It is very well possible to have a more general structure instead, namely directed acyclic graphs (DAGs) as this appendix explains. Such structures are encountered in an example in Section 5. In Figure 16 and example of such a structure is depicted..

The requirement that the classification is a directed tree is too restricted, as is shown here. A DAG is all that is required for a classification to be consistent. In this case a node may have more than one parent node. A DAG, or rather its closure, corresponds to a partial ordering structure.

**Figure 16. A directed acyclic graph.**



A DAG $G = (V, A)$, with $V$ the set of nodes and $A$ the set of arcs, still admits an additivity structure, that is, there is a function $f : A \rightarrow \Re$ that obeys Kirchhoff's condition

$$\sum_{b\in\{(u,v)\varepsilon A|u\varepsilon V\}} f(b) = \sum_{c\in\{(v,w)\varepsilon A|w\varepsilon V\}} f(c),$$

i.e. total inflow in node $v \in V$ equals total outflow of node $v \in V$, for any node $v \in V$ for which there is both inflow and outflow. From such a function f defined on the arcs we define a function *g* on its nodes, as follows:

$$g(v) = \sum_{b\varepsilon\{(w,v)\in A|w\varepsilon V\}} f(b),$$

for all nodes except a source node, i.e. a node with no parents, or

$$g(v) = \sum_{c \varepsilon \{(v,w) \in A | w \varepsilon V\}} f(c),$$

for all nodes $v$ except a sink node, i.e. a node with no children. For nodes $v$ with both parents and children the two definitions agree, so that $g : V \rightarrow \Re$ is well-defined, that is without ambiguity, due to Kirchhoff's condition, that applies to $f$. A function like $f$ can be viewed as a flow function. It defines a flow through $G$, although the flow values, that is, the values at the arcs, may be negative.

## Appendix B. Exceptions

We consider exceptions (or exclusions) in this appendix. In fact there are two places where we have to deal with them: firstly when processing descriptions and secondly when describing categories in a classification. We discuss both problem in this appendix, although briefly. The main goal is to call attention to the problem and describe a few of the problems involved.

Handling exceptions and exclusions is of importance in coding, although it is a somewhat specialised topic. The problem of recognizing an exception in a description is a linguistic problem. It requires parsing the description and analyzing it grammatically in order to identify it. The topic does not fit well within the approach of the current paper, as this approach does not use syntactical analysis methods, except tokenization. This may be a reason to consider extending our approach to automatic coding.

Exceptions can be indicated in many ways, in ordinary speech. Examples of words or phrases indicative of them are: ‘except’, ‘without’, ‘excluded’, ‘excluding’, ‘not including’, ‘not included’, ‘with the exclusion of’, etc.

Exceptions appear in descriptions as well as in code descriptions in the classification employed. Some of them should be considered bad practice. Examples from code descriptions are ‘all the remaining cases’ or ‘other, not mentioned elsewhere’.

**Example B.1.** What is the problem with exceptions like ‘Selling cloths, except children’s cloths’ in a business activity? A normal human being usually has no problems understanding such phrases, provided he or she understand the language in which they are phrased. It is a description, not by summing up its contents, but by first indicating an activity (‘selling’) of a bigger whole (‘cloths’) and then indicating the part that is to be excluded from this (‘children’s cloths’). ■

A description with an exception is comparable to a difference set, like $A \setminus B$. Using exceptions is sometimes a more convenient way of communicating than using descriptions without exclusions. The idea is that first a bigger set of possible events, objects, etc. is mentioned, after which some of these are excluded. We can illustrate this by the way sets can be defined. Take for instance the set $\{1,2,3,4,5,7,9,10\}$. This set can also be represented as $\{1,2,3,4,5,6,7,8,9,10\} \setminus \{6,8\}$. This uses a larger set, and takes away a subset. This way of describing the original set is obviously not unique. Instead one could also take

$\{1,2,3,4,5,6,7,8,9,10,538,1092\} \setminus \{6,8,538,1092\}$ to define the same set, but the choice is weird, and also not efficient because some elements are included in the bigger set that also are excluded in the ‘exclusion set’. The representation given before is more economical.

In principle it would be possible to describe the activity in Example B.1 in a positive way, by listing all the types of cloths that are being sold. The obvious drawback of this approach is that this is likely to be a quite long list of items, and one that even may not be complete. So it is more effective to use exceptions.

The easiest way to obtain exceptions is by asking explicitly for them in a separate question or set of questions. If this is not used exception should be recognized within the descriptions. Then descriptions should be scanned for certain keywords. As soon as an exception is identified, the next thing to do is to detect what words belong to it; there may several words in an exception.

In a coding system it may occur that some codes correspond to a remainder category, indicated with ‘other’. A particular code might be split up in a number of well-delimited categories, but they may not cover the entire code; some cases are left, and they are collected in such a remainder category. So such a code is attached to a description of the parent code applies to it and none of the other child codes.

## Appendix C. Long, composite words

In some languages (such as German, Dutch, Finnish) there is a possibility to produce words by concatenating several words or word forms. In that way long, composite words are produced. Long words are prone to spelling errors: they tend (in Dutch, at least) to be written as separate words under the influence of English.[5] But also the reverse might happen: words that should be separated are combined into a single word. Without special precautions against such errors, there may be problems recognizing the words intended by the respondents.

The process described below should only be applied in case part of a description seems to be free of D-words.

Syllabification of the words in a (corrected) description may be the first step to handle this problem. This is the process of dividing words into syllables. This yields the description as a string of syllables, with the interpunction and spaces retained. In the next step the spaces are discarded and an attempt is made to form words from consecutive syllables. In this process interpunction should be taken into account: syllables separated by certain interpunction symbols (such as dot, comma, semicolon, colon, etc) can never be combined as they are considered to belong to different sentences or parts of a sentence. This process is not guaranteed to lead to a unique segmentation of a description into words. Of course, it is a good idea to try to keep as close to the original sentence as possible.

Instead of starting with a syllabified description, it is also possible to see if syllabified D-words in the semantic network can be recognized in the description. The recognized parts should not overlap. The less syllables remain that are not recognized the better it is. This problem is an optimization problem.

Another way to handle the problem discussed in this appendix is by using trigrams instead of syllables.

[5] In The Netherlands this phenomenon is sometimes referred to as the 'English disease'

# Appendix D. Glossary

**Table D.1 Glossary for this paper**

| **Concept** | **Explanation** |
|---|---|
| Antonym | A semantic relation, denoting words of opposite meaning. Examples: light-dark, big-small, full-empty. |
| Category (in a classification) | A category can be characterized by a set of concepts. In fact there may be several sets of concepts that characterize the same category. Note that for each characterization the order of the concepts is unimportant |
| Classification | This can be represented as a finite directed tree. The nodes correspond to codes and the arcs indicate a parent-child relationship. |
| Code | If we view a classification as a tree structure, the codes correspond to the vertices (points) in the tree. In this paper each code from a certain level downwards may be the outcome of a coding process. |
| Desired level of coding | For a coding process, using a particular classification, one should specify the desired level of the target code. This means that codes on or below this level are acceptable, in the sense that they carry enough information. In case they are above that level of detail they are considered not precise enough and they require further information to be able to code the answer at the desired level of detail. Answers that are too detailed can be (uniquely) replaced by an answer that is at the desired level of detail, i.e. the coding level. See also 'Target code'. |
| Concept | A concept in the context of this paper is a (finite) set of C-words. Note that the order of these words is immaterial. |
| C-word | A word used to describe a category in a classification. Such a description is a (finite) set of C-words. C-words can be considered the formal part in a semantic network. C-words may actually belong to a different language than the remaining D-words. |
| D-word | A D-word is a word that is actually used in descriptions, possibly after they have been cleansed by a spell-checker. So they are supposed to be part of the vocabulary of respondents. They are all supposed to be connected to a specific domain, that is the subject of a survey or an inquiry. D-words include syntactical variations of words as well as words that will be discarded in the coding process ('stop words'). |
| ETL | Extract Transform Load. A step performed to make third party data useful for processing in one's own environment. Originally from the data warehousing area. Data cleansing is another expression used for this step. |
| Exception | 1. A way of specifying a state of affairs by first indicating a bigger set and then excluding some elements. It corresponds to the exclusion operation in set theory. Example: $A \setminus B$ which is the set consisting of the elements in the set A except those that also belong to the set B.<br>2. A situation in a process flow when a precondition for a step in the happy flow, is not met. |

| | |
|---|---|
| Happy flow | A flow through a process network in the ideal case, i.e. when all preconditions of each process step are met. |
| Holonym | A role in a semantic relation, which is the opposite of a meronym. This relationship indicates a part - whole opposition. The holonym is the whole side, and the meronym the part side of the relationship. Examples tree - wood, person - people, organ - body, engine – car. |
| Homonym | A semantic relationship, where a word (or concept) has two or more meanings. Examples: company, plant, mouth, mill, stream, desert, etc. |
| Hyperonym | A role in a semantic relation, which is the counterpart of that of hyponym. This relationship indicates a general-special opposition. The hyperonym indicates the more general part of this relation, the hyponym the more special part. Examples: fruit – apple, apple - Orange Pippin |
| Hyponym | See hyperonym |
| Learning (in a semantic network) | This is the ability of a semantic network to incorporate new knowledge, usually on the basis of new coding information. The basic input consists of new pairs of description-code combinations, or new D-words for which it has to be decided which of them link to C-words. Through learning, a semantic network can be expected to grow, because new D-words are added and by adding vertices/points and/or arcs. |
| Meronym | See holonym. |
| Semantic network (for coding) | In this paper a semantic network for coding can be viewed as a graph. The building blocks are words, concepts and codes, that form the set of vertices and the relations that exist among them that form the arcs. |
| Stop word | A D-word that does not carry enough information to be useful in coding, and that is, for that reason, not linked to a C-word. |
| Synonym | In this paper a semantic relationship between words or concepts that indicates similarity in meaning. It also does not distinguish between word forms such as verbs, nouns, adjectives, etc and also not between tenses (for verbs) and plurality or singularity (for nouns). Important is that these words share the same root. In linguistics such words are said to belong to the same word family |
| Target code | Code in a classification that is a potential output for a coding process using this classification. |
| Token | In the context of this paper a word in a description. |
| Tokenization | The process of dividing a description into tokens or words. |
| Transitivity | A binary relation R is transitive if for all a, b and c in this set, if aRb and bRc holds then also aRc holds. |
| Transitive closure | A binary relation R we can extend to a binary relation $R^*$ as follows:<br>1. for any a, b such that aRb holds, then define $aR^*b$ to hold<br>2. for any a, b, c such that aRb and bRc both hold, .then define $aR^*c$ to hold.<br>Then $R^*$ is the transitive closure of R. |
| Trigger | A description S triggers a C-word if S contains a D-word that is related to this C-word. A description triggers a concept if S contains D-words that are all related to all C-words in the concept. A description S triggers a code if S |

| | |
|---|---|
| | contains D-words that trigger a combination of concepts that define the code. Synonym: activate. |
| Uninformative word | See stop word |