LOGICA TRIANGULI

ŁÓDŹ, NANTES, SANTIAGO DE COMPOSTELA

4

2000

CONTENTS

Published yearly by the
University of Nantes, France

LOGICA TRIANGULI

is edited by

Patrice BAILHACHE (managing editor), Centre de Logique
Département de Philosophie, rue de la Censive du Tertre
BP 81227, 44312 NANTES CEDEX 3, France.
e-mail: patrice.bailhache@humana.univ-nantes.fr

Grzegorz MALINOWSKI, Department of Logic, University of Lódź, ul. Matejki 34a, 90-237 ŁÓDŹ, Poland.
e-mail: gregmal@krysia.uni.lodz.pl

Luis VILLEGAS-FORERO, Departamento de Lóxica e Filosofia da Ciencia, Filosofia do Dereito, Moral e Politica
15706 SANTIAGO DE COMPOSTELA, España.
e-mail: lflpvill@usc.es

The first purpose of *Logica Trianguli* is a presentation of the research in logic by members of the three following universities (the "Triangle"): in Lódź (Poland), in Nantes (France) and in Santiago de Compostela (Spain).

The other aim of the journal is to make known the results of the work related to the co-operation. *Logica Trianguli* is open to publish response and further discussion on the topics connected.

Papers for publication within the area of interest of the journal are invited. They should be written in English and conform the conditions indicated on the third page of the cover and may be sent to any of the persons indicated above.

LOGICA TRIANGULI

LOGIC IN ŁÓDŹ, NANTES, SANTIAGO DE COMPOSTELA

2000

Published yearly by the
University of Nantes, France

CONTENTS

Logica Trianguli, 4, 2000, p. 25-39

PROLOG FOR AUTOMATIC PROCESSING OF SYNONYMY*

Santiago FERNÁNDEZ LANZA

Abstract

In this work we propose a Prolog program that analyses the degree of synonymy of two words in a dictionary of synonyms, and their behaviour when they are substituted in a sentence.

1. Introduction

If we aim to characterise synonymy just as a human conceives it, we will have to consider aspects such as its contextuality, subjectivity, ambiguity, imprecision or gradual character. Faced with this variety and complexity of aspects involved, this work is aimed at an empirical study of synonymy rather than a theoretical one. This is based on the analysis of a short fragment of a dictionary of synonyms, and in the present work *The New Collins Dictionary and Thesaurus* has been used [3]. In the cited work, information regarding synonymy is structured in the following manner: word, list of synonymous words, meaning. For example, the word "fruit" has the list of synonyms {"crop", "harvest", "produce", "product", "yield"} in the first meaning, and {"advance", "benefit", "consequence", "effect", "outcome", "profit", "result", "return", "reward"} in the second.

The usual objective in using a dictionary of this type is to search for the synonyms of a word in order to substitute it for another in the sentence. However, if we consider synonymy as a question of degree, the new word may not mean exactly the same as the one replaced, and this may affect the informative content of the sentence as a whole.

To define a term a dictionary of synonyms gives a simple list of words. Thus, in order to verify whether two words are related, two lists of

* This work has been supported by the Xunta de Galicia project PGIDT99PXI10502B.

words can be compared. If synonymy is a matter of degree, and the meaning of a word is represented as the list of its synonyms, the question now is how to measure the degree of synonymy between two words. The treatment of natural language in Prolog by means of lists enables us to find this degree mechanically.

In the context of a dictionary of synonyms it is possible to consider different aspects related with this theme, which will be borne in mind throughout this work:

1. The problem of polysemy: Words may have various meanings depending on the context in use. This means that we are able to compare each word taking the generic set of its synonyms or only specific meanings, which are suitable for a particular use. This proposal appears to be more opportune, since it avoids the interference of other meanings in the decision of the synonymy of two words when they are used in a specific context.
2. The reflexivity of synonymy: Dictionaries of synonyms obviously do not include the trivial case of a word being a synonym of itself when a meaning is set. Nevertheless, this fact may lead to situations such as this one: in a dictionary of synonyms a word can appear as the only synonym of another and this word as the only synonym of the first. If we compare both words, the intersection of both sets of synonyms is empty, thus the degree of synonymity measured with the formula given earlier would be 0, a counterintuitive result. One way of avoiding this is to include the trivial case of the identical word in the list of synonyms of a word.
3. The symmetry of synonymy: It is common to attribute the property of symmetry to synonymy, but if we observe a dictionary of synonyms, in many cases this property does not hold; i.e., sometimes a word *A* has another *B* in its list of synonyms, but inverse case is not so.

2. Synonymy between words

In the Natural Language Processing tradition [1], we will describe a Prolog program which calculates the degree of synonymy between the words of a dictionary of synonyms. The examples are taken from [3]. Measurement is realised meaning by meaning, in such a manner that given two words *A* and *B*, the compiler will set the meaning of *A* verifying that *B* belongs to the list of synonyms of *A* in that meaning, it will calculate the degree of synonymy that *A* possesses with all the meanings of *B* and will select the meaning of *B* whose degree of synonymy is the greatest.

A dictionary of synonyms can be considered as a database of words (entries) and lists of words (synonyms) grouped together by meanings. We will use the following predicate

```
syn_dic(Word, List_of_synonyms, Meaning)
```

to represent the information contained in the dictionary. The first argument of the predicate `syn_dic` is the dictionary entry, the second the list of its synonyms and the third, the meanings corresponding to the group of synonyms. Let us consider an example that includes various input items from [3]:

```
syn_dic(fruit,[fruit,crop,harvest,produce,product,yield],1).
syn_dic(fruit,[fruit,advantage,benefit,consequence,effect,outcome,
profit,result,return,reward],2).

syn_dic(crop,[crop,fruit,gathering,harvest,produce,reaping,'season´s_
growth',vintage,yield],1).
syn_dic(crop,[crop,clip,curtail,cut,lop,mow,pare,prune,reduce,shear,
shorten,snip,top,trim],2).
syn_dic(crop,[crop,bring_home,bring_in,collect,garner,gather,harvest,
mow,pick,reap],3).
syn_dic(crop,[crop,browse,graze,nibble],4).

...
```

The general procedure for calculating the degree of synonymy between two words consists of the following steps. Given two words *A* and *B*:

1. Take the list of synonyms of *A* in the first meaning, which we will call *ListA*.
2. Check that *B* belongs to *ListA*.
   2.1. If it does not belong to *ListA*, carry out step 1 with the following meaning of *A*. If all the meanings of *A* have been checked and the result is still negative, then *A* and *B* are not synonyms.
   2.2. If it belongs to *ListA*, go to 3.
3. Take the list of synonyms of *B* in the first meaning (called *ListB*) and calculate the degree using one of the following formulae:

Jaccard coefficient:

$$DS = \frac{|ListA \quad ListB|}{|ListA \lceil \quad ListB|}$$

Dice coefficient:

$$DS = \frac{2|ListA \quad ListB|}{|ListA| + |ListB|}$$

Cosine coefficient:

$$DS = \frac{|ListA \quad ListB|}{\sqrt{|ListA|}\sqrt{|ListB|}}$$

Mutual similarity coefficient:

$$DS = \frac{\frac{|ListA \quad ListB|}{|ListA|} + \frac{|ListA \quad ListB|}{|ListB|}}{2}$$

Overlap coefficient:

$$DS = \frac{|ListA \quad ListB|}{min(|ListA|, |ListB|)}$$

4. Repeat step 3 with all the meanings of *B*.
5. Calculate the maximum of the degrees obtained in 4 and its corresponding meaning. This maximum degree is the degree of synonymy between *A* and *B*.

The predicate `synonym` calculates the degree of synonymy between two words:

```
synonym(A, MA, B, MB, DS, TH, SM)
```

this predicate can be read as, the word *A* in the meaning *MA* is a synonym of the word *B* in the meaning *MB* with a degree *DS* for the threshold *TH* and for the similarity measure *SM*.

The code is as follows:

```
synonym(A,X,A,Y,1.0,_,jaccard_coefficient):-
syn_dic(A,Z,X),
X=Y.
synonym(A,MeanA,B,MeanB,DS_MAX,TH,jaccard_coefficient):-
comparable(A,B,MeanA),
findall(GRS,syn_meaning(A,MeanA,B,Mean,GRS,TH,jaccard_coefficient),Li
st1),
findall(Mean,syn_meaning(A,MeanA,B,Mean,GRS,TH,jaccard_coefficient),L
ist2),
max(List2,MeanB,List1,DS_MAX).

synonym(A,X,A,Y,1.0,_,dice_coefficient):-
syn_dic(A,Z,X),
X=Y.
synonym(A,MeanA,B,MeanB,DS_MAX,TH,dice_coefficient):-
comparable(A,B,MeanA),
findall(GRS,syn_meaning(A,MeanA,B,Mean,GRS,TH,dice_coefficient),List1
),
findall(Mean,syn_meaning(A,MeanA,B,Mean,GRS,TH,dice_coefficient),List
2),
max(List2,MeanB,List1,DS_MAX).

synonym(A,X,A,Y,1.0,_,cosine_coefficient):-
syn_dic(A,Z,X),
X=Y.
synonym(A,MeanA,B,MeanB,DS_MAX,TH,cosine_coefficient):-
comparable(A,B,MeanA),
findall(GRS,syn_meaning(A,MeanA,B,Mean,GRS,TH,cosine_coefficient),Lis
t1),
findall(Mean,syn_meaning(A,MeanA,B,Mean,GRS,TH,cosine_coefficient),Li
st2),
max(List2,MeanB,List1,DS_MAX).
```

```
synonym(A,X,A,Y,1.0,_,mutual_similarity_coefficient):-
syn_dic(A,Z,X),
X=Y.
synonym(A,MeanA,B,MeanB,DS_MAX,TH,mutual_similarity_coefficient):-
comparable(A,B,MeanA),
findall(GRS,syn_meaning(A,MeanA,B,Mean,GRS,TH,mutual_similarity_coeff
icient),List1),
findall(Mean,syn_meaning(A,MeanA,B,Mean,GRS,TH,mutual_similarity_coef
ficient),List2),
max(List2,MeanB,List1,DS_MAX).

synonym(A,X,A,Y,1.0,_,overlap_coefficient):-
syn_dic(A,Z,X),
X=Y.
synonym(A,MeanA,B,MeanB,DS_MAX,TH,overlap_coefficient):-
comparable(A,B,MeanA),
findall(GRS,syn_meaning(A,MeanA,B,Mean,GRS,TH,overlap_coefficient),Li
st1),
findall(Mean,syn_meaning(A,MeanA,B,Mean,GRS,TH,overlap_coefficient),L
ist2),
max(List2,MeanB,List1,DS_MAX).
```

The predicate `comparable` enables us to verify whether a word is in the list of synonyms of another one, once the specific meaning of the latter has been set. Thus

`comparable(A, B, MA)`

can be read as: word *A* is comparable with another *B* in the particular meaning *MA* of *A*.

```
comparable(A,B,MA):-
 syn_dic(A,X,MA),
 member(B,X).
```

The predicate `member` is normally used in the treatment of lists with Prolog, and indicates if a particular element is a member of a list:

`member(Element, List)`

In Prolog this can be expressed in the following manner:

```
member(A,[A|B]).
member(A,[B|C]):-
 member(A,C).
```

The maximum of the degrees with the corresponding meaning is calculated by means of the predicate

`max(List_of_meanings, Meaning, List_of_degrees, Greatest_degree).`

The first argument is a list of meanings, the second the meaning corresponding to the greatest degree, the third is a list of degrees, and the fourth the greatest of these degrees.

max([B],B,[A],A).

```
max([D,E|F],Y,[A,B|C],X):-
 A >= B,
 max([D|F],Y,[A|C],X),
 !.
max([D,E|F],Y,[A,B|C],X):-
 max([E|F],Y,[B|C],X).
```

The predicate `syn_meaning` calculates the degree of synonymy for the meanings of two specific words

`syn_meaning(A, MA, B, MB, DS, TH, SM)`

It is necessary to define this predicate for all the similarity measures.

1.- *Jaccard coefficient:*

```
syn_meaning(A,MA,B,MB,DS,TH,jaccard
_coefficient):-
 not(A=B),
 syn_dic(A,X,MA),
 syn_dic(B,Y,MB),
 union(X,Y,U),
 inter(X,Y,I),
 card(U,NU),
 card(I,NI),
 DS is NI/NU,
 DS >= TH.
```

2.- *Dice coefficient*:

```
syn_meaning(A,MA,B,MB,DS,TH,dice_
coefficient):-
 not(A=B),
 syn_dic(A,X,MA),
 syn_dic(B,Y,MB),
 inter(X,Y,I),
 card(I,NI),
 card(X,NX),
 card(Y,NY),
 DS is (2*NI)/(NX+NY),
 DS >= TH.
```

3.- *Cosine coefficient*:

```
syn_meaning(A,MA,B,MB,DS,TH,cosine_
coefficient):-
 not(A=B),
 syn_dic(A,X,MA),
 syn_dic(B,Y,MB),
 inter(X,Y,I),
 card(I,NI),
 card(X,NX),
 card(Y,NY),
 SQRX is sqrt(NX),
 SQRY is sqrt(NY),
 DS is NI/(SQRX*SQRY),
 DS >= TH.
```

4.- *Mutual similarity coefficient*:

```
syn_meaning(A,MA,B,MB,DS,TH,mutua
l_similarity_coefficient):-
 not(A=B),
 syn_dic(A,X,MA),
 syn_dic(B,Y,MB),
 inter(X,Y,I),
 card(I,NI),
 card(X,NX),
 card(Y,NY),
 SLR is NI/NX,
 SRL is NI/NY,
 DS is (SLR+SRL)/2,
 DS >= TH.
```

5.- *Overlap coefficient*:

```
syn_meaning(A,MA,B,MB,DS,TH,overlap_coefficient):-
 not(A=B),
 syn_dic(A,X,MA),
 syn_dic(B,Y,MB),
 inter(X,Y,I),
 card(I,NI),
 card(X,NX),
 card(Y,NY),
 min(NX,NY,Min),
```

```
 DS is NI/Min,
 DS >= TH.
```

The predicate `union` establishes the union of the two lists and has three lists as arguments:

union(List1, List2, Result)

The code is as follows:

```
union([],X,X).
union([A|B],Y,[A|Z]):-
 not(member(A,Y)),
 union(B,Y,Z),
 !.
union([A|B],Y,Z):-
 union(B,Y,Z).
```

The predicate `inter` establishes the intersection of the two lists and also has three lists as arguments:

inter(List1, List2, Result)

The code is as follows:

```
inter([],X,[]).
inter([A|B],Y,[A|Z]):-
 member(A,Y),
 inter(B,Y,Z),
 !.
inter([A|B],Y,Z):-
 inter(B,Y,Z).
```

The predicate `card` indicates the number of elements in a list. Its first argument is a list and the second one is a natural number that corresponds to the cardinality of the list:

card(List, Cardinality)

The code is as follows:

```
card([],0).
card([A|B],C):-
 card(B,E),
 C is E + 1.0.
```

2.1. Verbalisation of the degree of synonymy between words

As has already been seen, the degree of synonymy is expressed as a real number between 0 and 1. However, to a user a linguistic verbalisation may be easier to understand than a degree. One immediate and simple manner of doing this is by using a predicate that is analogous to the one previously explained, but which instead of giving a degree of synonymy gives a verbalisation:

syn_verb(A, MA, B, MB, Verb, TH, SM)

where *A* and *B* are words, *MA* and *MB* are its respective meanings, *Verb* is the linguistic label that corresponds to the degree of synonymy, *TH* is the threshold and *SM* is the similarity measure.

```
syn_verb(A,MA,A,MA,identical_words,TH,SM):-
 synonym(A,MA,B,MB,1,TH,SM).
syn_verb(A,MA,B,MB,Verb,TH,SM):-
 synonym(A,MA,B,MB,DS,TH,SM),
 not(A=B),
 verbal(DS,Verb).
```

The predicate `verbal` associates a verbalisation to a particular degree of synonymy,

```
verbal(Degree_of_synonymy, Verbalisation)
```

Its first argument is a degree (real number between 0 and 1) and the second one is the expression that verbalises this degree. The criterion for the distribution of the intervals of each verbalisation is arbitrary; the only thing that is aimed for with this is to establish a reasonable procedure so that the compiler's reply be more attractive. To this aim, the following interval distribution is proposed:

1. All degrees greater than 0 and lower than or equal to 0.25 are verbalised with the label "very little synonymy".
2. All degrees greater than 0.25 and lower than or equal to 0.5 with "little synonymy".
3. All degrees greater than 0.5 and lower than or equal to 0.75 with "quite a lot of synonymy".
4. All degrees greater than 0.75 and lower than or equal to 1 with "a lot of synonymy".

The following code enables us to do this:

```
verbal(Degree,very_little_synonymy):-
 Degree > 0,
 Degree =< 0.25.
verbal(Degree,little_synonymy):-
 Degree > 0.25,
 Degree =< 0.5.
verbal(Degree,quite_a_lot_of_synonymy):-
 Degree > 0.5,
 Degree =< 0.75.
verbal(Degree,a_lot_of_synonymy):-
 Degree > 0.75,
 Degree =< 1.0.
```

2.2. Examples of synonymy between words

We can now ask what the degree of synonymy is between two words belonging to the dictionary, such as "fruit" and "produce", with the threshold 0 an for the Jaccard coefficient. A goal of this kind is formulated in the following manner:

```
synonym(fruit,Meaning_fruit,produce,Meaning_produce,Degree,0,jaccard_
coeficient).
```

As is usual in Prolog, the words in capitals are taken as variables that the program must instantiate on responding. The compiler would respond with:

```
Meaning_fruit = 1
Meaning_produce = 7
Degree = 0.625
```

With the predicate `syn_verb` this degree would be verbalised as "quite_a_lot_of_synonymy".

We will be able to ask which words are synonyms of "fruit" with the threshold 0 and for the Jaccard coefficient. This goal is formulated as follows:

```
synonym(fruit, Meaning_fruit, Y, Meaning_Y, Degree, 0,
jaccard_coefficient).
```

The compiler answers with all the synonyms of the word fruit indicating the meanings of both words and the degree of synonymy between them.

3. Synonymy between sentences

If we conceive synonymy as a question of degree, how does the substitution of one word for its synonym affect the degree of synonymy of two sentences? We now attempt to establish some proposals that refer to this question.

Two sentences are synonyms with a degree of synonymy *DS* if their words coincide or are synonyms respectively. When two sentences differ in two or more words and these are synonyms, the degree of synonymy of both sentences will be the result of operating the degrees of synonymy of these words using a t-norm, respectively. In this work, product, minimum and Łukasiewicz t-norms will be used.

Considering a sentence as a list, which is usual in the treatment of natural language using Prolog, the predicate `syn_sent` expresses the synonymy between two sentences:

```
syn_sent(Sentence1, Sentence2, Degree_of_synonymy, M1, M2, TH, SM,
TN)
```

The first two arguments are lists in which two sentences from natural language are included, the third is the degree of synonymy between both sentences, the fourth is a list with the words substituted in the first sentence followed by the corresponding meanings. The fifth argument is the same as the fourth except that it has the list of the substitutions in the second sentence. The three last arguments are the threshold, the similarity measure and the t-norm respectively.

This predicate will be defined for each t-norm:

1. *Product*:

```
syn_sent([],[],1.0,[],[],_,_,product).
syn_sent([A|B],[A|D],X,Y,Z,TH,SM,product):-
 syn_sent(B,D,X,Y,Z,TH,SM,product).
syn_sent([A|B],[C|D],Y,[A,Mean_A|W],[C,Mean_C|Q],TH,SM,product):-
 synonym(A,Mean_A,C,Mean_C,Z,TH,SM),
 syn_sent(B,D,X,W,Q,TH,SM,product),
 Y is X*Z,
 Y >= TH.
```

2. *Minimum*:

```
syn_sent([],[],1.0,[],[],_,_,minimo).
syn_sent([A|B],[A|D],X,Y,Z,TH,SM,minimo):-
 syn_sent(B,D,X,Y,Z,TH,SM,minimo).
syn_sent([A|B],[C|D],Y,[A,Mean_A|W],[C,Mean_C|Q],TH,SM,minimo):-
 synonym(A,Mean_A,C,Mean_C,Z,TH,SM),
 syn_sent(B,D,X,W,Q,TH,SM,minimo),
 min(X,Z,Y),
 Y >= TH.
```

3. *Łukasiewicz*:

```
syn_sent([],[],1.0,[],[],_,_,lukasiewicz).
syn_sent([A|B],[A|D],X,Y,Z,TH,SM,lukasiewicz):-
 syn_sent(B,D,X,Y,Z,TH,SM,lukasiewicz).
syn_sent([A|B],[C|D],Y,[A,Mean_A|W],[C,Mean_C|Q],TH,SM,lukasiewicz):-
 synonym(A,Mean_A,C,Mean_C,Z,TH,SM),
 syn_sent(B,D,X,W,Q,TH,SM,lukasiewicz),
 SUM is X+Z-1,
 maxim(0,SUM,Y),
 Y >= TH.
```

The predicate `maxim` calculates the maximum of two numbers. It is defined in the following manner:

```
maxim(A,B,A):-
 A>=B,
 !.
maxim(A,B,B).
```

3.1. Synonymy between sentences setting the meanings

When we use words in a sentence we usually do so with a specific meaning (except if we are using the language in a different way from the common usage, e.g., literary use). Although, this is not always the case, the sentence usually gives us hints about which of the meanings of a particular word is being used. On other occasions we need other types of extralinguistic information in order to elucidate this problem.

Various criteria can be put forward for detecting which meaning of the word is used in a sentence. These criteria should not be too strict as they are not applicable to all cases. Some of these may be the following:

1. If in the same sentence there are two words that are synonyms respectively in particular meanings, it is very probable that these meanings will be the ones used in the sentence.
2. Very often, certain words, which do not need to be synonyms of a given one, indicate that this word is being used in a particular meaning. For example, the word "bank" in English has various meanings, such as "financial institution", "lateral inclination", "ground beside a river", etc.; but if in a particular sentence it appears together with words such as "money", "cheque", "cash", "loan", etc. this indicates that "bank" is being used in the first of the meanings.

3.1.1. First criterion for setting meanings

In order to deal with the first criterion we will use the following predicate:

synonym_sent(Sentence1, Sentence2, Degree_of_synonymy, M1, M2, TH, SM, TN)

This predicate is formed by the same arguments as `syn_sent`, but taking into account that the existence of certain words in a sentence may set the meanings of others.

The code is as follows:

```
synonym_sent(A,B,GR,Z,Y,TH,SM,TN):-
 meaning_sent_sent(A,A,Z),
 syn_sent(A,B,GR,Z,Y,TH,SM,TN),
 not(Z = []),
```

```
 not(Y = []).
synonym_sent(A,B,GR,W,Y,TH,SM,TN):-
 meaning_sent_sent(A,A,Z),
 Z = [],
 syn_sent(A,B,GR,W,Y,TH,SM,TN),
 not(W = []),
 not(Y = []).
```

The predicate `meaning_sent_sent` is the one which enables us to verify whether all the words of one sentence can be associated to all those of another one. Eventually, in our case, this second sentence will be the same as the first one, although this may not be the case if we wish to set the meanings of the words of one sentence with regard to other sentences different from the one given. The predicate has three lists as arguments:

meaning_sent_sent(Sentence1, Sentence2, List_of_meanings)

the first two lists include two sentences (eventually the same one) and in the third one words followed by the set meanings.

The code is the following:

```
meaning_sent_sent([],X,[]).
meaning_sent_sent([A|B],X,[R,S|Y]):-
 meaning_word_sent(A,X,Z),
 not(Z=[]),
 remove_rep(Z,[R,S]),
 meaning_sent_sent(B,X,Y),
 !.
meaning_sent_sent([A|B],X,[A,K|Y]):-
 meaning_word_sent(A,X,Z),
 not(Z=[]),
 remove_rep(Z,W),
 not(card(W,2)),
 meaning_sent_sent(B,X,Y),
 !.
meaning_sent_sent([A|B],X,Y):-
 meaning_sent_sent(B,X,Y).
```

Given a word and a sentence, the predicate `meaning_word_sent`, gives a list of the meanings of the word that result from associating it with the words in the sentence. The result is the initial word followed by the meaning used in the association, this being repeated as many times as associations have been made. The arguments of this predicate are one word, one list in which we will include a sentence, and another list which will include the word with the meanings resulting from the associations realised.

meaning_word_sent(Word, Sentence, List_of_meanings)

```
meaning_word_sent(A,[],[]).
meaning_word_sent(A,[B|C],[A,MA|E]):-
 not(A=B),
 associate(A,B,MA),
```

```
 meaning_word_sent(A,C,E),
 !.
meaning_word_sent(A,[B|C],D):-
 meaning_word_sent(A,C,D).
```

When in the sentence there are various words associable with the first one for the same meaning, in the last list the meaning will appear as many times as associations have been made. In order to avoid repetition of meanings, we will use the predicate `remove_rep`, the arguments of which are lists:

```
remove_rep(List_with_repetitions, List_without_repetitions)
```

The first argument is the list with the repeated meanings, and the second is the list with no repeated meanings.

```
remove_rep([],[]).
remove_rep([A,MA,B,MB|C],[B,MB|D]):-
 [A,MA]=[B,MB],
 remove_rep([B,MB|C],[B,MB|D]),
 !.
remove_rep([A,MA|C],[A,MA|D]):-
 remove_rep(C,D).
```

The predicate `associate` used in `meaning_word_sent` enables the program to associate words. The arguments of this predicate are two words and a meaning corresponding to the first word:

```
associate(A, B, MA)
```

```
associate(A,B,MA):-
 synonym(A,MA,B,MB,DS,0,_);
 synonym(B,MB,A,MA,DS,0,_).
```

3.1.2. Second criterion for setting meanings

In order to report on the second of these criteria that set the meanings in a sentence, it is necessary to create another database that associates particular words with other ones to set the meanings. One way could be to consider that words such as "farmer" may indicate that the first meaning of "fruit" is being used, and words such as "enjoy" may indicate that the second one is being used. We could then create a new database clauses like the following:

```
associate(fruit,farmer,1).
associate(fruit,farmers,1).
...
associate(fruit,enjoy,2).
associate(fruit,enjoyed,2).
...
```

It is clear that any conjugation of the verb "enjoy" in any tense, mood, number and person will be valid. The predicates `remove_rep`, `meaning_word_sent`, `meaning_sent_sent` and `synonym_sent` are analogous to the ones defined for the previous criterion. The predicate `associate` defined there must be substituted for the above-mentioned database.

3.2. Verbalisation of the degree of synonymy between sentences

As was carried out for synonymy between words, we can define a predicate that instead of supplying a numerical degree of synonymy, gives a verbalisation of it. This is the following:

```
synonym_sent_verb(Sentence1, Sentence2, Verbalisation, M1, M2)
```

This predicate is made up of two list sentences, the verbalisation of the degree of synonymy between these sentences, and two lists that contain the set meanings in the first and the second sentence, respectively.

The code is as follows:

```
synonym_sent_verb(A,B,Verb,W,Y,TH,SM,TN):-
 synonym_sent(A,B,DS,W,Y,TH,SM,TN),
 verbal(DS,Verb).
```

3.3. Examples of synonymy between sentences

Given a threshold, a similarity measure and a t-norm, we could ask which sentences are synonyms of "Our fruit or crop is the best", This goal is formulated as follows:

```
synonym_sent([our,fruit,or,crop,is,the,best],B,Degree,M1,M2,0,jaccard_co
efficient,product).
```

The program sets meaning 1 of "fruit" and meaning 1 of "crop" due to both words appearing in the sentence and being associable in these meanings. The compiler answers with all the synonym sentences indicating the degree of synonymy and the meaning of the substituted words.

Using the code defined for the second criterion we can attempt the following goal:

```
synonym_sent([he,enjoyed,the,fruit,(s),of,the,previous,years,work],B,
Degree,M1,M2,0,jaccard_coefficient,product).
```

In this case, the compiler will set meaning 2 of "fruit".

4. Conclusions

We have proposed a Prolog program that analyses the degree of synonymy between the words of a dictionary of synonyms and how the

substitution of synonymous words in a sentence may affect the degree of synonymy between the original sentence and that in which the word has been substituted.

As frequently occurs when dealing with problems relating to natural language, these matters have a domain so extensive that may provoke fluctuations regarding the results offered in this work. Other functions may be used to measure the degree of synonymy; new criteria to set the meanings in synonymy between sentences or other interval distributions for verbalisation may be adopted. In any case, synonymy is a theme that has been studied relatively little in the setting of Prolog and the aim of this work has been to explain only certain aspects relating to its computational processing.

University of Santiago de Compostela, sflanza@usc.es

REFERENCES

[1] COVINGTON, M.A., *Natural Language Processing for Prolog Programmers*, Prentice-Hall, 1994.

[2] LÓPEZ DE MÁNTARAS, R., TRILLAS, E., "Towards a measure of the degree of synonymy", *in* Sánchez, E. (ed.), *Fuzzy Information, Knowledge Representation and Decision Analysis*, Pergamon Press, 1984.

[3] MCLEOD, W.T. (ed.), *The New Collins Dictionary and Thesaurus in One Volume*, Collins, 1989.

[4] SPARCK JONES, K., *Synonymy and Semantic Classification*, Edinburgh U. P., 1986.

N° 1 (1997)

PATRICE BAILHACHE, Modal Logic: A-Completeness Met Up Again - ANDRZEJ INDRZEJCZAK, Generalised Sequent Calculus for Propositional Modal Logics - PIOTR LUKOWSKI, The Nature of Intuitionistic possibility - GRZEGORZ MALINOWSKI, On Many-Valuedness, Sentential Identity, Inference and Lukasiewicz Modalities - CONCHA MARTINEZ, JOSE-MIGUEL SAGÜILLO, JAVIER VILANOVA, Fitch's Problem and The Knowability Paradox : Logical and Philosophical Remarks - DOROTA RYBARKIEWICZ, Three Aspects of Metaphor - FRANÇOIS SCHMITZ, Modal Logic and the "Possible" - LUIS VILLEGAS-FORERO, JANUSZ MACIASZEK, Tarski on Logical Entities

N° 2 (1998)

PATRICE BAILHACHE, How to Mix Alethic, Deontic, Temporal, Individual Modalities - ANTONIO BLANCO SALGUEIRO, Holism and Dependency among Properties - JEAN-LOUIS GARDIES, Basic Logic for Ontic and Deontic Modalities - ALAIN LECOMTE, Multimodal Logic for Syntax - PIOTR LUKOWSKI, The Law of Excluded Middle and Intuitionistic Logic - MAREK NOWAK, Kripke Semantics for Some Paraconsistent Logics - DOROTA RYBARKIEWICZ, The Structure of Metaphor towards a Pragmatic Approach - JAVIER VILANOVA ARIAS, Natural Language Conditionals - ALEJANDRO SOBRINO, JOSE ANGEL OLIVAS, SANTIAGO FERNANDEZ, Semi-sentences, Semi-strings and Semi-grammatical Rules in Prolog

N° 3 (1999)

JANUSZ CIUCIURA, History and Development of the Discursive Logic - JOSE L. FALGUERA, Ontosemantic Divergence and Comparability of Theories - ANDRZEJ INDRZEJCZAK, A Survey of Natural Deduction Systems for Modal Logics - SERGE LAPIERRE, FRANÇOIS LEPAGE, Completeness and Representation Theorem for Epistemic States in First-order Predicate Calculus - GRZEGORZ MALINOWSKI, Formalization of Intensional Functions and Epistemic Knowledge Representation Systems - UXIA RIVAS MONROY, Proper Names: One Century of Discussion

ISBN 2-913276-03-2

9 782913 276031