Pattern Matching for Case Analysis: A Computational Definition of Closeness *

Sylvain Delisle, Terry Copeck, Stan Szpakowicz, Ken Barker

Department of Computer Science, University of Ottawa
Ottawa, Ontario, Canada K1N 6N5
Sylvain_Delisle@uqtr.uquebec.ca
{terry, szpak, kbarker}@csi.uottawa.ca

Abstract

We propose a conceptually and technically neat method to identify known semantic patterns close to a novel pattern. This occurs in the context of a system to acquire knowledge incrementally from systematically processed expository technical text. This semi-automatic system requires the user to respond to specific multiple-choice questions about the current sentence. The questions are prepared from linguistic elements previously encountered in the text similar to elements in the new sentence. We present a metric to characterize the similarity between semantic Case patterns. The computation is based on syntactic indicators of semantic relations and is defined in terms of symbolic pattern matching.

1. Introduction

1.1. The need for a closeness computation

The TANKA project (Text ANalysis for Knowledge Acquisition) seeks to build a model of a technical domain by semi-automatically processing unedited English text that describes the domain. Sentences are parsed and patterns of the Cases associated with its verbs are extracted from the parse. Concepts derived from these Case structures are added to a growing network that represents knowledge about the domain. The DIPETT parser within TANKA has a very broad coverage of English syntax. HAIKU, the module which uses the closeness metric presented in this paper, performs user-assisted semantic interpretation. It finds phrases that represent Cases in DIPETT's parse tree and uses its past processing experience to select the most likely Case realizations of each phrase using little *a priori* semantic knowledge. The user validates these selections and the system interactively learns new Case patterns as it goes through the input text.

One mechanism essential to HAIKU is a procedure to produce syntactic Case Marker patterns close to the pattern in the input sentence. It is invoked whenever the input does not match any previously processed pattern.

The TANKA project is a long-term research initiative of the Knowledge Acquisition Lab at our Department. Previously we presented the overall design [6], discussed aspects of the knowledge representation used in TANKA [5,7,8], and described the parser and the HAIKU Case Analyzer [4,3,2].

1.2. Basic definitions

Case Analysis and the closeness computation work on patterns of Case markers (syntactic) and Cases (semantic) stored in the following data structures:

Case Marker Pattern (CMP): an ordered list of Case Markers from a single clause in the input sentence. A Case Marker is a preposition, a noun's position[1] or an adverb. It is the linguistic representation of an argument to the verb. For example, the clause "The program executed at midnight" is associated with the CMP `psubj-at`, where `psubj` marks the subject "the program" and `at` marks the prepositional phrase "at midnight". The CMP representing "They executed the program daily" is `psubj-pobj-adv`, where `psubj` marks the subject "they", `pobj` marks the object "the program" and `adv` marks the adverb "daily".

Case Pattern (CP): an ordered list of Cases from a single clause in the input sentence. For example, the CP associated with "The program executed at midnight" is AGT-TAT, where AGT (agent) is the Case associated with "the program" and TAT (time at) is the Case associated with "at midnight". The list of Cases used in TANKA, as well as an extensive list of Case markers, can be found in [1].

1.3. Case analysis in HAIKU

The Case Analyzer begins by searching its databases for the CP that exactly or most closely matches the CP of the input clause. It then looks for an example sentence to illustrate this matching CP. If the input is parsable, only one CP should be associated with a given input after user interaction and approval.

In certain situations the Case Analyzer must rank partially-matching CMPs in terms of their closeness to the input CMP. The algorithm to compute the closeness of two patterns is based on notions of under-matching and overmatching among CMPs. It is similar in spirit to some methods used in pattern recognition for computing distances between patterns [9]. A CMP is a set, represented here as a Prolog list e.g. `[psubj,pobj,at]`. Intuitively, an undermatch occurs when a given CMP is completely or partially contained in another CMP. The computation of closeness is described in Prolog; predicates not defined here come from the Quintus Prolog library.

2. Definitions of matching

Pattern S_1 can match pattern S_2 in three ways, listed here in the order of decreasing discrimination from the perspective of Case Analysis: it can match perfectly (PM), as a proper subset (PUM), and via a non-empty intersection set (NPUM). Disjoint patterns are never considered as matching.

2.1. PM: perfect match

Two patterns have identical membership: $S_1 \equiv S_2$.

EXAMPLES:

```
If S1=[psubj,pobj,at] and S2=[psubj,pobj,at],
then S1 and S2 match perfectly.
If S1=[psubj,pobj] and S2=[psubj,pobj,at],
then S1 and S2 do not match perfectly.
```

IMPLEMENTATION:

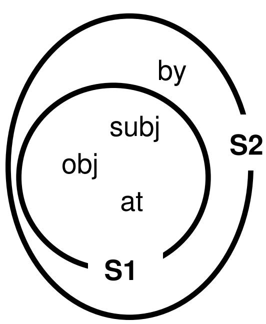
```
% seteq: set equality
pm(S1, S2) :- seteq(S1, S2).
```

2.2. PUM: proper undermatch

S_2 is larger than S_1. We say that S_1 properly undermatches S_2, or that S_2 properly overmatches S_1.

EXAMPLES:

```
If S1=[psubj,pobj,at] and S2=[psubj,pobj,at],
then proper undermatch fails.
If S1=[psubj,pobj,at] and
S2=[psubj,pobj,at,by], then proper undermatch
succeeds. See figure below.
If S1=[psubj,pobj,at,by] and
S2=[psubj,pobj,by,from], then proper
undermatch fails.
```

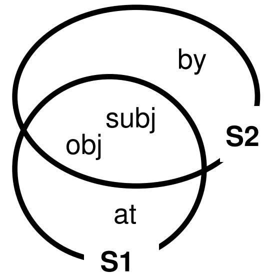


IMPLEMENTATION:

```
% subset(S1, S2): S1 is a subset of S2
pum(S1, S2) :- \+ seteq(S1, S2),
      subset(S1, S2).
```

2.3. NPUM: non-proper undermatch

The patterns are different, but there is a non-empty intersection between them. Cardinality (technically, the length of a list) is used as a criterion to emphasize specificity: a higher cardinality indicates a more discriminating set. (Notice that the npum relation is symmetric if the lengths of the patterns are the same.)



EXAMPLES:

```
If S1 = [psubj,pobj,at] and S2 =
[psubj,pobj,by], then S1 non-properly
undermatches S2. See figure above.
If S1 = [psubj,adv] and S2 = [pobj,by,from],
then non-proper undermatch fails.
If S1 = [psubj,pobj,at,from] and S2 =
[pobj,at,adv], then non-proper undermatch fails
since S2 is shorter than S1.
```

IMPLEMENTATION:

```
npum(S1, S2) :-
      \+ subset(S1, S2),
      length(S1, LS1),  length(S2, LS2),
      LS1 =< LS2,
      intersection(S1, S2, ITST),
      ITST \== [].
```

3. Basic computations for a closeness metric

The following procedures refine those outlined in section 2. Additional parameters return numerical values of closeness factors. The new versions supply zeroes instead of failing when a match is not found.

IMPLEMENTATION:

```
pum(S1, S2, CF) :-
      \+ seteq(S1, S2), subset(S1, S2), !,
      length(S1, LS1), length(S2, LS2),
      CF is LS1/LS2.
pum(_S1, _S2, 0).
```

3.1. Computing the closeness factor within pum

The closeness factor CF measures how much S_1 differs from S_2. It allows us to compare any two patterns S_1 and S_2 quantitatively: *the higher the CF, the closer the patterns are to one another.*

EXAMPLES:

```
If S1 = [psubj,pobj] and S2 = [psubj,pobj,at],
then CF = 2 / 3.
If S1 = [psubj,pobj] and S2 =
[psubj,pobj,at,from], then CF = 2 / 4.
If S1 = [psubj,pobj,at] and S2 =
[psubj,pobj,at,from], then CF = 3 / 4.
```

3.2. Computing the comparison coefficient within npum

In this definition, CF_{12} and CF_{21} are the closeness factors between S_1 and S_2 calculated from the perspective of S_1 and S_2 respectively. CC, the Comparison Coefficient between them, is computed separately.

IMPLEMENTATION:

```
npum(S1, S2, CF12, CF21, CC) :-
   \+ subset(S1, S2),
   intersection(S1, S2, ITST), ITST \== [],
   !,  length(S1, LS1),  length(S2, LS2),
   LS1 =< LS2,
   length(ITST, N),  CF12 is N/LS1,
   CF21 is N/LS2,
   compute_CC(ITST, CF12, CF21, CC).
npum(_S1, _S2, 0, 0, 0).
```

The greater CF_{ij}, the closer the intersection ITST is to S_i. Since S_2 can be no smaller than S_1 (LS1 ≤ LS2)[2], CF_{21} must always be equal to or smaller than CF_{12}, which can never equal 1. Also, the nature of CMs in the intersection is significant. More importance should be attached to ITST = `[pobj]` than to ITST = `[adv]`: for the vast majority of English verbs, an object is semantically more informative than an adverb. The calculation of the Comparison Coefficient takes these facts into account.

The Comparison Coefficient represents the syntactico-semantic similarity between two situations expressed in two more or less similar clauses. CC is the product of CF_{12}, CF_{21} and V, a number that depends on the specific nature of the markers in the intersection: *the more significant these markers are, in terms of Case Analysis, the higher V should be*. CF_{12} and CF_{21} are multiplied to obtain the highest ranking for patterns closest to the input pattern. Addition might be used instead of multiplication. Multiplying numbers from the interval [0, 1] resembles finding the joint probability of independent events. It is a matter of experiment to see whether using addition would drastically change the behavior of the `compute_CC` algorithm. CC is computed as follows.

IMPLEMENTATION:

```
compute_CC(ITST, CF12, CF21, CC) :-
      get_ind_values(ITST,
List_of_Values),
      sumlist(List_of_Values, Sum),
      CC is (CF12 * CF21 * Sum).
get_ind_values([], []) :- !.
get_ind_values([H | Hs], [V | Vs]) :-
      get_ind_value(H, V),
      get_ind_values(Hs, Vs).
```

The predicate `get_ind_value` supplies a numerical value for a Case Marker reflecting its semantic specificity in Case-based semantic analysis. The pattern `[psubj,pobj]` is less specific than `[psubj,pobj,at,from]` because it occurs with more verbs and so is less discriminating. A perfectly specific CMP would occur with one and only one verb.

Our table of CM values reflects a belief that subjects and objects are less semantically significant than prepositions but more significant than adverbs. The values are subjective but can be tailored to the type of text at hand or modified according to assumptions one may wish to make and verify on certain texts. Experimenting may help standardize such values. We will refer to this table as the *CM-ranking*.

IMPLEMENTATION:

```
get_ind_value(Prep,  2.5) :-
   dict(Prep,prep,_,_), !.
get_ind_value(piobj, 2.0).
get_ind_value(pobj,  1.5).
get_ind_value(psubj, 1.0).
get_ind_value(adv,   0.5).
```

3.3. Examples of closeness calculations

The four following examples assume that attempts to match by `pm(S1, S2)` and `pum(S1, S2, _)` fail but non-proper undermatching invoked by `npum(S1, S2, CF12, CF21, CC)` succeeds.

```
a)  Let S1 = [psubj,pobj,at] and S2  =
    [psubj,pobj,from], so that
    ITST=[psubj,pobj]
 CF12 = 2 / 3, CF21 = 2 / 3,
 CC = (1 + 1.5) * (2 / 3) * (2 / 3) = 1.11
```

```
b)  Let S1 = [psubj,pobj,at] and S2  =
    [psubj,pobj,from,adv], so that
    ITST=[psubj, pobj]
CF12 = 2 / 3, CF21 = 2 / 4,
CC = (1 + 1.5) * (2 / 3) * (2 / 4) = 0.83
```

According to the CM-ranking, S_1 and S_2 are closer to one another in a) than in b). Also `[psubj, pobj, at]` is intuitively closer to `[psubj,pobj,from]` than to `[psubj,pobj,from,adv]`: the added adverb in the second pattern diminishes the closeness of the two sets because it is not matched in the first set.

```
c)  Let S1 = [pobj,by,adv] and S2  =
    [psubj,pobj,adv], so that ITST=[pobj,adv]
CF12 = 2 / 3, CF21 = 2 / 3,
CC = (1.5 + 0.5) * (2 / 3) * (2 / 4) = 0.67
```

```
d)  Let S1 = [pobj,by,adv] and S2  =
    [psubj,pobj,by], so that ITST=[pobj,by]
CF12 = 2 / 3, CF21 = 2 / 3,
CC = (1.5 + 2.5) * (2 / 3) * (2 / 3) = 1.78
```

According to the CM-ranking, S_1 and S_2 are closer to one another in d) than in c). Two patterns which match on a particular preposition are more similar than those which match on an unspecified adverb.

4. Computational details of the closeness metric

In this section In_CMP refers to the pattern of the clause currently being processed by the system. L_in is the list of known CMPs in TANKA's dictionaries against which In_CMP will be matched. The closeness metric is only calculated when a perfect match between the input CMP and known CMPs cannot be found.

4.1. Top-level procedure to compute closeness

The procedure begins with a call to `find_closer_patterns/6`. This predicate takes In_CMP and L_in as inputs and returns four ordered lists of CMPs, L_out_1, L_out_2, L_out_3, L_out_4, that properly undermatch, non-properly undermatch, properly overmatch and non-properly overmatch In_CMP.

The order of these arguments in `find_closer_pattern` reflects HAIKU's preference in matches: proper undermatching, non-proper undermatching, proper overmatching, non-proper overmatching. Undermatching is preferable to overmatching. Undermatching provides the system with candidate patterns that express known meaning and subsume the input CMP. Overmatching introduces a greater uncertainty: candidate patterns here correspond to a potentially large class of underspecified situations that may represent semantically more distant candidate CMPs.

Suppose the input CMP is `[psubj,pobj,at]`, and the CMP dictionary contains `[psubj,pobj]` and `[psubj,pobj,at,from]`. The input CMP is a proper undermatch of `[psubj,pobj,at,from]` and a proper overmatch of `[psubj,pobj]`. The former is a much more discriminating candidate than `[psubj,pobj]`. If the user approves `[psubj,pobj,at,from]` the system can focus on a relatively small set of Case Patterns as candidates for the current input. Giving priority to `[psubj,pobj]` would force the user to deal with a much larger set of CPs and render the knowledge acquired interactively less certain.

Proper matching is preferred to non-proper matching. A proper match is more likely to represent a valid situation, whereas a non-proper match may correspond to a situation which could be semantically less fitting.

Each of the four output lists is ordered by decreasing specificity: longest matches are preferred to shortest ones because they correspond to closer neighbours of the input CMP in the space of known CMPs. The closer the neighbour, the more likely it is that it will lead to a valid candidate semantic pattern.

IMPLEMENTATION:

```
/* Given a list L_in of already known
CMPs, this predicate finds the lists (in
decreasing order of 'closeness') of
patterns in L_in that are close to
In_CMP, the input CMP. */
find_closer_patterns(
      In_CMP, L_in, L1, L2, L3, L4) :-
  best_Under_Match(In_CMP, L_in, L1, L2),
  best_Over_Match(In_CMP, L_in, L3, L4).
```

4.2. A procedure to find best undermatches

We now present the computational details of finding the "best neighbours" for a given input CMP (In_CMP). These candidate patterns are found in terms of best proper undermatches and best non-proper undermatches with the input CMP. The procedures are defined using 'pum' and 'npum' which were described earlier.

IMPLEMENTATION:

```
/* L_in : list of patterns against which
          In_CMP is tentatively matched.
   L_out: sorted list of pum/npum patterns
          (in decreasing order).
Elements of L_out have the form
'match(Closeness, Pattern)'.

best_Under_Match(
   In_CMP, L_in, L_out_pum, L_out_npum) :-
  best_under_match_pum(
     In_CMP, L_in, [], L_out_pum),
  best_under_match_npum(
     In_CMP, L_in, [], L_out_npum).

% PROPER UNDERMATCHING
best_under_match_pum(
  _S, [], L_in, L_out) :-
      sort(L_in, L_out_temp),
      reverse(L_out_temp, L_out).
best_under_match_pum(
  S, [S_H | T], L_in, L_out) :-
      pum(S, S_H, CF_H),  CF_H \== 0,  !,
      best_under_match_pum(
          S, T, [match(CF_H, S_H) | L_in],
          L_out).
best_under_match_pum(
  S, [_S_H | T], L_in, L_out) :-
      best_under_match_pum(
          S, T, L_in, L_out).

% NON-PROPER UNDERMATCHING
best_under_match_npum(
  _S, [], L_in, L_out) :-
      sort(L_in, L_out_temp),
      reverse(L_out_temp, L_out).
best_under_match_npum(
  S, [S_H | T], L_in, L_out) :-
      npum(S, S_H, _, _, CC_H), !,
      best_under_match_npum(
          S, T, [match(CC_H, S_H) | L_in],
          L_out).
best_under_match_npum(
  S, [S_H | T], L_in, L_out) :-
      best_under_match_npum(
          S, T, L_in, L_out).
```

4.3. A procedure to find best over-matches

The definitions of `best_over_match_pum` and `best_over_match_npum` are analogous to those of `best_under_match_pum` and `best_under_match_npum` with the first two parameters (`S` and `S_H`) inverted.

Note that `npum` has been defined with the condition "LS1 ≤ LS2". Substituting "LS1 < LS2" in the '`best_over_match`' versions eliminates patterns that would appear in both overmatching and undermatching sets.

```
IMPLEMENTATION:
best_Over_Match(
  In_CMP, L_in, L_out_pum, L_out_npum) :-
      best_over_match_pum(
          In_CMP, L_in, [], L_out_pum),
      best_over_match_npum(
          In_CMP, L_in, [], L_out_npum).
```

4.4. Examples

Two examples illustrate the computation of the closeness metric. Suppose Case Analysis has found a new CMP with a known verb. Assume perfect match and proper undermatch each fail in both examples. (A problem not discussed here is that of the ordering of Case Markers in CMPs: we assume a standard and consistent ordering of these patterns in the dictionaries and their related access routines.)

Example #1: "A system runs a datalink from Toronto before midnight."

```
    CMP:    psubj-pobj-before-from
The CMP is represented by S1=[psubj, pobj,
before, from]
find_closer_patterns(S1, Set_of_all_CMPs, L1,
L2, L3, L4) finds:
L1 = [ [psubj,pobj,before,from,adv] ]
(properly overmatching CMPs);
L2 = [
[psubj,pobj,at,before],[psubj,pobj,at,by]]
          (non-properly overmatching CMPs);
L3 = []                        (properly
undermatching CMPs);
L4 = []              (non-properly
undermatching CMPs);
```

The list of candidate CMPs sorted in descending order of closeness is: [psubj-pobj-before-from-adv, psubj-pobj-at-before, psubj-pobj-at-by]. psubj-pobj-before-from-adv would be the first CMP HAIKU suggests to the user. If it is rejected, psubj-pobj-at-before would then be suggested, and so on.

Examples of answers for L3 and L4 would be:

```
L3 = [ [psubj,pobj,before],
       [psubj,pobj,from],  ... ]
L4 = [ [psubj,pobj,at],
       [psubj,pobj,in], [psubj,pobj,adv],  ...
]
```

Example #2: "The system runs a datalink by 7 midnight."

```
    CMP:      psubj-pobj-by
The CMP is represented by S1=[psubj, pobj, by]
find_closer_patterns(S1, Set_of_all_CMPs, L1,
L2, L3, L4) finds:
L1 = [ [psubj,pobj,at,by] ] (properly
overmatching CMPs);
L2 = [ [psubj,pobj,by,from],
       [psubj,pobj,from], [psubj,pobj,adv] ]
    (non-properly overmatching CMPs);
L3 = [ [psubj,pobj] ]
                        (properly undermatching
CMPs);
L4 = []                (non-properly
overmatching CMPs);
```

The list of candidate CMPs sorted in descending order of closeness is: [psubj-pobj-at-by, psubj-pobj-by-from, psubj-pobj-from, psubj-pobj-adv, psubj-pobj]. psubj-pobj-at-by would be suggested first.

4.5. Strategic aspects of the closeness computation

The distinction between undermatching and overmatching is not important to the user when he/she looks at candidates. Find_closer_patterns/6 will return a correctly ordered list of potential candidates from both populations. Wasting time on unlikely candidate patterns could be avoided if CA uses a threshold for values of CF (for pum) and of CC (for npum) below which candidates are discarded. Alternatively, an upper limit could be set on the number of candidates to be considered: then if find_closer_patterns/6 returns more than one candidate, HAIKU might never go past the second or third.

5. Summary of the algorithm

The dictionary of Case Marker Patterns is represented in Figure 1 below as a lattice into which the heavily outlined input pattern has been inserted in the appropriate location. To simplify things non-proper overmatches, the least preferred type, are not shown.

The closeness algorithm can be summarized as follows, assuming a given CM-ranking:

- First find any undermatching patterns. Sort the list of results to have at its head matches with the most matching CMs and fewest unmatched CMs. Sort matches of the same length and number of matching CMs to favour the most heavily-weighted CMs.
- Next find any overmatching patterns. Sort the list of results to have at the head matches with the fewest additional CMs. Sort matches of the same length to bring forward those with the most heavily-weighted CMs.

find_closer_patterns/6 currently returns the CMP closest to the input CMP according to the following priorities:

properly undermatching CMPs
> non-properly undermatching CMPs
> properly overmatching CMPs
> non-properly overmatching CMPs.

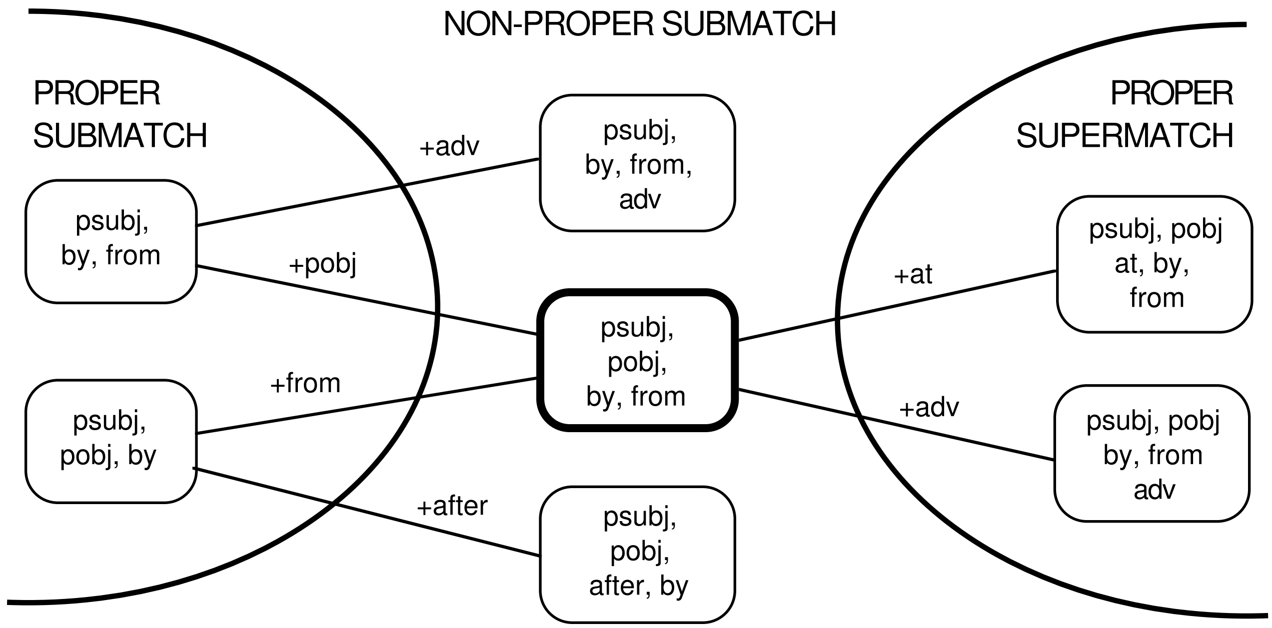


Figure 1. Relationships between Case Marker Patterns

6. Discussion

We have translated the difficult problem of finding the best approximation of a given semantic patterns into the computationally feasible task of finding the best match for a pattern of symbols which denote basic semantic relations. Linguistically, we propose to define semantic similarity via syntactic congruence and compute it from syntactic markers (the Case Marker Patterns). The metric we propose allows our system to suggest to the user valid candidate semantic patterns for the input at hand, based on the system's knowledge accumulated so far. The metric can be used in any setting in which such a substitution of syntactic for semantic phenomena can be justified.

A bibliographic search revealed a great deal of very distantly related work on pattern matching. Surprisingly, the problem presented here seems not to be recorded in the literature. We see two explanations. The approach pursued in TANKA (the primacy of syntax in knowledge acquisition from text) places the emphasis on syntactic substitutes for semantic criteria in a manner not required in other work. Or the problem itself may not be new but the setting where it was encountered may be, requiring an innovative solution that should be efficient and linguistically justifiable.

References

[1] Barker, K., T.Copeck, S. Delisle, & S. Szpakowicz (1993) "An Empirically Grounded Case System", TR-93-08, Department of Computer Science, University of Ottawa.

[2] Delisle, S., K.Barker, T.Copeck, & S.Szpakowicz (1992), "Interactive Semantic Analysis in TANKA: Designing HAIKU", TR-92-32, Department of Computer Science, University of Ottawa.

[3] Copeck, T., S. Delisle & S. Szpakowicz (1992) "Parsing and Case Analysis in TANKA". *Proc 15th Intl Conf on Computational Linguistics COLING-92*, Nantes, July 1992, 1008-1012.

[4] Delisle, S. & S. Szpakowicz (1991) "A Broad-Coverage Parser for Knowledge Acquisition from Technical Texts". E. Johnson (ed.) *Proc Fifth Int Conf on Symbolic and Logical Computing*, Dakota State University, 169-183.

[5] Szpakowicz, S. & Z.Koperczak (1990) "Mixed-Strategy Matching in Conceptual Networks". Z. W. Ras, M. Zemankova & M. L. Emrich (eds.), *Methodologies for Intelligent Systems 5*. North-Holland, 321-328.

[6] Szpakowicz, S. (1990) "Semi-automatic Acquisition of Conceptual Structure from Technical Texts". *Int J Man-Machine Studies*, **33**, 385-397.

[7] Yang, L. & S.Szpakowicz (1991a) "Inheritance in Conceptual Networks". Karen S. Harber (ed.) *Proc Sixth Int Symposium on Methodologies for Intelligent Systems (Poster Session)*. Charlotte, NC, Oct. 1991, 191-202.

[8] Yang, L. & S.Szpakowicz (1991b) "Planning in Conceptual Networks". F.Dehne, F.Fiala & W.W.Koczkodaj (eds.) *Advances in Computing and Information - ICCI '91. Lecture Notes in Computer Science,* 497, Springer-Verlag, 669-671.

[9] Belaïd, A. & Y. Belaïd (1992), *Reconnaissance des Formes (méthodes et applications)*, InterÉditions.

---

* This work is supported by the Natural Sciences and Engineering Research Council of Canada.

[1] The three positional Case markers are the syntactic subject (psubj), object (pobj), and indirect object (piobj).

[2] In fact, npum has two versions. One, npum_u, is used by best_under_match_npum and uses the test LS1 ≤ LS2. The other, npum_o, is used by best_over_match_npum and uses LS1 < LS2.