SOLVING ANALOGIES ON WORDS: AN ALGORITHM

Yves Lepage
ATR Interpreting Telecommunications Research Labs,
Hikaridai 2-2, Seika-tyō, Sōraku-gun, Kyōto 619-0288, Japan
lepage@itl.atr.co.jp

Introduction

To introduce the algorithm presented in this paper, we take a path that is inverse to the historical development of the idea of analogy (see (Hoffman 95)). This is necessary, because a certain incomprehension is faced when speaking about linguistic analogy, *i.e.*, it is generally given a broader and more psychological definition. Also, with our proposal being computational, it is impossible to ignore works about analogy in computer science, which has come to mean artificial intelligence.

1 A Survey of Works on Analogy

This paper is not intended to be an exhaustive study. For a more comprehensive study on the subject, see (Hoffman 95).

1.1 Metaphors, or Implicit Analogies

Beginning with works in psychology and artificial intelligence, (Gentner 83) is a milestone study of a possible modeling of analogies such as, "*an atom is like the solar system*" adequate for artificial intelligence. In these analogies, two domains are mapped, one onto the other, thus modeling of the domain becomes necessary.

$$\texttt{sun} \xrightarrow{f} \texttt{nucleus}$$
$$\texttt{planet} \xrightarrow{f} \texttt{electron}$$

In addition, properties (expressed by clauses, formulae, *etc.*) are transferred from one domain onto the other, and their number somehow determines the quality of the analogy.

$$\texttt{attracts(sun, planet)} \xrightarrow{f} \texttt{attracts(nucleus, electron)}$$
$$\texttt{moremassive(sun, planet)} \xrightarrow{f} \texttt{moremassive(nucleus, electron)}$$

However, Gentner's explicit description of sentences as "*an A is like a B*" as analogies is subject to criticism. Others (*e.g.* (Steinhart 94)) prefer to call these sentences *metaphors*[1], the validity of which rests on sentences of the kind, "*A is to B as C is to D*", for which the name *analogy*[2] is reserved. In other words, some metaphors are supported by analogies. For instance, the metaphor, "*an atom is like the solar system*", relies on the analogy, "*an electron is to the nucleus, as a planet is to the sun*".[3]

The answer of the AI community is complex because they have headed directly to more complex problems. For them, in analogies or metaphors (Hall 89):

- two different domains appear
- for both domains, modeling of a knowledge-base is necessary
- mapping of objects and transfer of properties are different operations
- the quality of analogies has to be evaluated as a function of the strength (number, truth, *etc.*) of properties transferred.

We must drastically simplify all this and enunciate a simpler problem (whose resolution may not necessarily be simple). This can be achieved by simplifying data types, and consequently the characteristics of the problem.

[1]If the fact that properties are carried over characterises such sentences, then etymologically they are metaphors: In Greek, *pherein*: to carry; *meta-*: between, among, with, after. "Metaphor" means to transfer, to carry over.

[2]In Greek, *logos, -logia*: ratio, proportion, reason, discourse; *ana-*: top-down, again, anew. "Analogy" means the same proportions, similar ratios.

[3]This complies with Aristotle's definitions in the Poetics.

1.2 Multiplicity *vs* Unicity of Domains

In the field of natural language processing, there have been plenty of works on pronunciation of English by analogy, some being very much concerned with reproducing human behavior (see (Damper & Eastmond 96)). Here is an illustration of the task from (Pirelli & Federici 94):

$$\begin{tikzcd}
\textit{vane} \arrow[r, "f"] \arrow[d, "g"] & /\text{vejn}/ \arrow[d, "h"] \\
\textit{sane} \arrow[r, "f"] & x = /\text{sejn}/
\end{tikzcd}$$

Similarly to AI approaches, two domains appear (graphemic and phonemic). Consequently, the functions f, g and h are of different types because their domains and ranges are of different data types.

Similarly to AI again, a common feature in such pronouncing systems is the use of data bases of written and phonetic forms. Regarding his own model, (Yvon 94) comments that:

> The [...] model crucially relies upon the existence of numerous *paradigmatic relationships* in lexical data bases.

Paradigmatic relationships being relationships in which four words intervene, they are in fact morphological analogies: "reaction *is to* reactor, *as* faction *is to* factor".

$$\begin{tikzcd}
\textit{reactor} \arrow[r, "f"] \arrow[d, "g"] & \textit{reaction} \arrow[d, "g"] \\
\textit{factor} \arrow[r, "f"] & \textit{faction}
\end{tikzcd}$$

Contrasting sharply with AI approaches, morphological analogies apply in only one domain, that of words. As a consequence, the number of relationships between analogical terms decreases from three (f, g and h) to two (f and g). Moreover, because all four terms intervening in the analogy are from the same domain, the domains and ranges of f and g are identical. Finally, morphological analogies can be regarded as simple equations independent of any knowledge about the language in which they are written. This standpoint eliminates the need for any knowledge base or dictionary.

$$\begin{tikzcd}
\textit{reactor} \arrow[r, "f"] \arrow[d, "g"] & \textit{reaction} \arrow[d, "g"] \\
\textit{factor} \arrow[r, "f"] & x?
\end{tikzcd}$$

1.3 Unicity *vs* Multiplicity of Changes

Solving morphological analogies remains difficult because several simultaneous changes may be required to transform one word into a second (for instance, *doer* $\rightarrow$ *undo* requires the deletion of the suffix *-er* and the insertion of the prefix *un-*). This problem has yet to be solved satisfactorily. For example, in (Yvon 94), only one change at a time is allowed, and multiple changes are captured by successive applications of morphological analogies (*cascade* model). However, there are cases in the morphology of some languages where multiple changes at the same time are mandatory, for instance in semitic languages.

"One change at a time", is also found in (Nagao 84) for a translation method, called *translation by analogy*, where the translation of an input sentence is an adaptation of translations of similar sentences retrieved from a data base. The difficulty of handling multiple changes is remedied by feeding the system with new examples differing by only one word commutation at a time. (Sadler and Vendelmans 90) proposed a different solution with an algebra on trees: differences on strings are reflected by adding or subtracting trees. Although this seems a more convincing answer, the use of data bases would resume, as would the multiplicity of domains.

Our goal is a true analogy-solver, *i.e.*, an algorithm which, on receiving three words as input, outputs a word, analogical to the input. For that, we thus have to answer the hard problem of: (1) performing multiple changes (2) using a unique data-type (words) (3) without dictionary nor any external knowledge.

1.4 Analogies on Words

We have finished our review of the problem and ended up with what was the starting point of our work. In linguistic works, *analogy* is defined by Saussure, after Humboldt and Baudoin de Courtenay, as the operation by which, given two forms of a given word, and only one form of a second word, the missing form is coined[4], "*honor* is to *honōrem* as *ōrātor* is to *ōrātōrem*" noted *ōrātōrem* : *ōrātor* = *honōrem* : *honor*. This is the same definition as the one given by Aristotle himself, "*A is to B as C is to D*", postulating identity of types for *A*, *B*, *C*, and *D*.

[4] Latin: *ōrātor* (orator, speaker) and *honor* (honour) nominative singular, *ōrātōrem* and *honōrem* accusative singular.

However, while analogy has been mentioned and used, algorithmic ways to solve analogies seem to have never been proposed, maybe because the operation, is so "intuitive". We (Lepage & Ando 96) recently gave a tentative computational explanation which was not always valid because false analogies were captured. It did not constitute an algorithm either.

The only work on solving analogies on words seems to be Copycat ((Hofstadter et al. 94) and (Hoffman 95)), which solves such puzzles as: $abc : abbccc = ijk : \mathtt{x}$. Unfortunately it does not seem to use a truly dedicated algorithm, rather, following the AI approach, it uses a formalisation of the domain with such functions as, "`previous in alphabet`", "`rank in alphabet`", *etc.*

2 Foundations of the Algorithm

2.1 The First Term as an Axis

(Itkonen and Haukioja 97) give a program in Prolog to solve analogies in sentences, as a refutation of Chomsky, according to whom analogy would not be operational in syntax, because it delivers non-grammatical sentences. That analogy would apply also to syntax, was advocated decades ago by Hermann Paul and Bloomfield. Chomsky's claim is unfair, because it supposes that analogy applies only on the symbol level. Itkonen and Haukioja show that analogy, when controlled by some structural level, does deliver perfectly grammatical sentences. What is of interest to us, is the essence of their method, which is the seed for our algorithm:

> Sentence D is formed by going through sentences B and C one element at a time and inspecting the relations of each element to the structure of sentence A (plus the part of sentence D that is ready).

Hence, sentence A is the axis against which sentences B and C are compared, and by opposition to which output sentence D is built.

$$\mathit{reader} : \underline{\mathit{un}}\mathit{read}\underline{\mathit{able}} = \overline{\mathit{do}}\mathit{er} : \mathtt{x} \Rightarrow \mathtt{x} = \underline{\mathit{un}}\overline{\mathit{do}}\underline{\mathit{able}}$$

The method will thus be: (*a*) look for those parts which are not common to A and B on one hand, and not common to A and C on the other and (*b*) put them together in the right order.

2.2 Common Subsequences

Looking for common subsequences of A and B (resp. A and C) solves problem (*a*) by complementation. (Wagner & Fischer 74) is a method to find longest common subsequences by computing edit distance matrices, yielding the minimal number of edit operations (insertion, deletion, substitution) necessary to transform one string into another.

For instance, the following matrices give the distance between *like* and *unlike* on one hand, and between *like* and *known* on the other hand, in their right bottom cells: dist(*like*, *unlike*) = 2 and dist(*like*, *known*) = 5

$$\begin{array}{c|cccccc} & u & n & l & i & k & e \\ \hline l & 1 & 2 & 2 & 3 & 4 & 5 \\ i & 2 & 2 & 3 & 2 & 3 & 4 \\ k & 3 & 3 & 3 & 3 & 2 & 3 \\ e & 4 & 4 & 4 & 4 & 3 & 2 \end{array} \qquad \begin{array}{c|ccccc} & k & n & o & w & n \\ \hline l & 1 & 2 & 3 & 4 & 5 \\ i & 2 & 2 & 3 & 4 & 5 \\ k & 2 & 3 & 3 & 4 & 5 \\ e & 3 & 3 & 4 & 4 & 5 \end{array}$$

2.3 Similitude between Words

We call *similitude* between A and B the length of their longest common subsequence. It is also equal to the length of A, minus the number of its characters deleted or replaced to produce B. This number we call pdist(A, B), because it is a pseudo-distance, which can be computed exactly as the edit distances, except that insertions cost 0.

$$\mathrm{sim}(A, B) = \mid A \mid - \,\mathrm{pdist}(A, B)$$

For instance, pdist(*unlike*, *like*) = 2, while pdist(*like*, *unlike*) = 0.

$$\begin{array}{c|cccc} & l & i & k & e \\ \hline u & 1 & 1 & 1 & 1 \\ n & 2 & 2 & 2 & 2 \\ l & 2 & 2 & 2 & 2 \\ i & 3 & 2 & 2 & 2 \\ k & 4 & 3 & 2 & 2 \\ e & 5 & 4 & 3 & 2 \end{array} \qquad \begin{array}{c|cccccc} & u & n & l & i & k & e \\ \hline l & 1 & 1 & 0 & 0 & 0 & 0 \\ i & 2 & 2 & 1 & 0 & 0 & 0 \\ k & 3 & 3 & 2 & 1 & 0 & 0 \\ e & 4 & 4 & 3 & 2 & 1 & 0 \end{array}$$

Characters inserted into B or C may be left aside, precisely because they are those characters of B and C, absent from A, that we want to assemble into the solution, D.

As A is the axis in the resolution of analogy, graphically we make it the vertical axis around which the computation of pseudo-distances takes place. For instance, for *like* : *unlike* = *known* : x,

$$\begin{array}{ccccc|c|cccccc} n & w & o & n & k & & u & n & l & i & k & e \\ \hline 1 & 1 & 1 & 1 & 1 & l & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 & i & 2 & 2 & 1 & 0 & 0 & 0 \\ 2 & 2 & 2 & 2 & 2 & k & 3 & 3 & 2 & 1 & 0 & 0 \\ 3 & 3 & 3 & 3 & 3 & e & 4 & 4 & 3 & 2 & 1 & 0 \end{array}$$

2.4 The Coverage Constraint

It is easy to verify that there is no solution to an analogy if some characters of A appear neither in B nor in C. The contrapositive says that, for an analogy to hold, any character of A has to appear in either B or C. Hence, the sum of the similitudes of A with B and C must be greater than or equal to its length: $\text{sim}(A,B)+\text{sim}(A,C) \geq |A|$, or, equivalently,

$$|A| \geq \text{pdist}(A,B)+\text{pdist}(A,C)$$

When the length of A is greater than the sum of the pseudo-distances, some subsequences of A are common to all strings in the same order. Such subsequences have to be copied into the solution D. We call $\text{com}(A,B,C,D)$ the sum of the length of such subsequences. The delicate point is that this sum depends precisely on the solution D being currently built by the algorithm.

To summarise, for analogy $A : B = C : D$ to hold, the following constraint must be verified:

$$|A| = \text{pdist}(A,B)+\text{pdist}(A,C)+\text{com}(A,B,C,D)$$

3 The Algorithm

3.1 Computation of Matrices

Our method relies on the computation of two pseudo-distance matrices between the three first terms of the analogy. A result by (Ukkonen 85) says that it is sufficient to compute a diagonal band plus two extra bands on each of its sides in the edit distance matrix, in order to get the exact distance, if the value of the overall distance is known to be less than some given threshold. This result applies to pseudo-distances, and is used to reduce the computation of the two pseudo-distance matrices. The width of the extra bands is obtained by trying to satisfy the coverage constraint with the value of the current pseudo-distance in the other matrix.

```
proc compute_matrices($A, B, C, pd_{AB}, pd_{AC}$)
    compute pseudo-distances matrices with
    extra bands of $pd_{AB}/2$ and $pd_{AC}/2$
    if $|A| \geq \text{pdist}(A,B)+\text{pdist}(A,C)$
        main component
    else
        compute_matrices($A, B, C,$
            $\max(|A| - \text{pdist}(A,C), pd_{AB}+1),$
            $\max(|A| - \text{pdist}(A,B), pd_{AC}+1)$)
    end if
end proc compute_matrices
```

3.2 Main Component

Once enough in the matrices has been computed, the principle of the algorithm is to follow the paths along which longest common subsequences are found, simultaneously in both matrices, copying characters into the solution accordingly. At each time, the positions in both matrices must be on the same horizontal line, *i.e.* at a same position in A, in order to ensure a right order while building the solution, D.

Determining the paths is done by comparing the current cell in the matrix with its three previous ones (horizontal, vertical or diagonal), according to the technique in (Wagner & Fischer 74). As a consequence, paths are followed from the end of words down to their beginning. The nine possible combinations (three directions in two matrices) can be divided into two groups: either the directions are the same in both matrices, or they are different.

The following sketches the algorithm. $\text{com}(A,B,C,D)$ has been initialised to: $|A| - (\text{pdist}(A,B) + \text{pdist}(A,C))$. i_A, i_B and i_C are the current positions in A, B and C. dir_{AB} (resp. dir_{AC}) is the direction of the path in matrix $A \times B$ (resp. $A \times C$) from the current position. "copy" means to copy a character from a word at the beginning of D and to move to the previous character in that word.

```
if constraint($i_A, i_B, i_C, \text{com}(A,B,C,D)$)
    case: $dir_{AB} = dir_{AC} =$ diagonal
        if $A[i_A] = B[i_B] = C[i_C]$
            decrement $\text{com}(A,B,C,D)$
        end if
        copy $B[i_B] + C[i_C] - A[i_A]$[a]
    case: $dir_{AB} = dir_{AC} =$ horizontal
        copy char[b]/ $\min(\text{pdist}(A[1..i_A], B[1..i_B]),$
                         $\text{pdist}(A[1..i_A], C[1..i_C]))$
    case: $dir_{AB} = dir_{AC} =$ vertical
        move only in $A$ (change horizontal line)
    case: $dir_{AB} \neq dir_{AC}$
        if $dir_{AB} =$ horizontal
            copy $B[i_B]$
```

[a]In this case, we move in the three words at the same time. Also, the character arithmetics factors, in view of generalisations, different operations: if the three current characters in A, B and C are equal, copy this character, otherwise copy that character from B or C that is different from the one in A. If all current characters are different, this is a failure.

[b]The word with less similitude with A is chosen, so as to make up for its delay.

```
        else if $dir_{AB}$ = vertical
            move in $A$ and $C$
        else same thing by exchanging $B$ and $C$
        end if
end if
```

3.3 Early Termination in Case of Failure

Complete computation of both matrices is not necessary to detect a failure. It is obvious when a letter in A does not appear in B or C. This may already be detected before any matrix computation.

Also, checking the coverage constraint allows the algorithm to stop as soon as non-satisfying moves have been performed.

3.4 An Example

We will show how the analogy *like* : *unlike* = *known* : x is solved by the algorithm.

The algorithm first verifies that all letters of *like* are present either in *unlike* or *known*. Then, the minimum computation is done for the pseudo-distances matrices, *i.e.* only the minimal diagonal band is computed.

$$\begin{array}{cccccccccccc}
e & k & i & l & n & u & & k & n & o & w & n \\
 & & & & & & & & & & & \\
. & . & . & 0 & 1 & 1 & l & 1 & 1 & . & . & . \\
. & . & 0 & 1 & 2 & . & i & . & 2 & 2 & . & . \\
. & 0 & 1 & 2 & . & . & k & . & . & 3 & 3 & . \\
0 & 1 & 2 & . & . & . & e & . & . & . & 4 & 4
\end{array}$$

As the coverage constraint is verified, the main component is called. It follows the paths noted by values in circles in the matrices.

$$\begin{array}{cccccccccccc}
e & k & i & l & n & u & & k & n & o & w & n \\
 & & & & & & & & & & & \\
. & . & . & \textcircled{0} & \textcircled{1} & \textcircled{1} & l & \textcircled{1} & \textcircled{1} & . & . & . \\
. & . & \textcircled{0} & 1 & 2 & . & i & . & 2 & \textcircled{2} & . & . \\
. & \textcircled{0} & 1 & 2 & . & . & k & . & . & 3 & \textcircled{3} & . \\
\textcircled{0} & 1 & 2 & . & . & . & e & . & . & . & 4 & \textcircled{4}
\end{array}$$

The succession of moves triggers the following copies into the solution:

dir_{AB}	dir_{AC}	copy
diagonal	diagonal	*n*
diagonal	diagonal	*w*
diagonal	diagonal	*o*
diagonal	diagonal	*n*
horizontal	horizontal	*k*
horizontal	diagonal	*n*
horizontal	diagonal	*u*

At each step, the coverage constraint being verified, finally, the solution x = *unknown* is ouptut.

4 Properties and Coverage

4.1 Trivial Cases, Mirroring

Trivial cases of analogies are, of course, solved by the algorithm, like: $A : A = A : \mathtt{x} \ \Rightarrow\ \mathtt{x} = A$ or $A : A = C : \mathtt{x} \ \Rightarrow\ \mathtt{x} = C$. Also, by construction, $A : B = C : \mathtt{x}$ and $A : C = B : \mathtt{x}$ deliver the same solution.

With this construction, mirroring poses no problem. If we note $\overline{A}$ the mirror of word A, then $A : B = C : D \Leftrightarrow \overline{A} : \overline{B} = \overline{C} : \overline{D}$.

4.2 Prefixing, Suffixing, Parallel Infixing

Appendix A lists a number of examples, actually solved by the algorithm, from simple to complex, which illustrate the algorithm's performance.

4.3 Reduplication and Permutation

The previous form of the algorithm does not produce *reduplication*. This would be necessary if we wanted to obtain, for example, plurals in Indonesian[5]: *orang* : *orang-orang* = *burung* : x $\Rightarrow$ x = *burung-burung* . In this case, our algorithm delivers, x = *orang-burung*, because preference is given to leave prefixes unchanged. However, the algorithm may be easily modified so that it applies repeatedly so as to obtain the desired solution[6].

Permutation is not captured by the algorithm. An example (*q* with *a* and *u*) in Proto-semitic is: *yaqtilu* : *yuqtilu* = *qatal* : *qutal*.

4.4 Language-independence/Code-dependence

Because the present algorithm performs computation only on a symbol level, it may be applied to any language. It is thus language independent. This is fortunate, as analogy in linguistics certainly derives from a more general psychological operation ((Gentner 83), (Itkonen 94)), which seems to be universal among human beings. Examples in Section A illustrate the language independence of the algorithm.

Conversely, the symbols determine the *granularity* of the analogies computed. Consequently, a commutation not reflected in the coding system will not be captured. This may be illustrated by a Japanese example in three different

[5] *orang* (human being) singular, *orang-orang* plural, *burung* (bird).

[6] Similarly, it is easy to apply the algorithm in a transducer-like way so that it modifies, by analogy, parts of an input string.

codings: the native writing system, the Hepburn transcription and the official, strict recommendation (kunrei).

Kanji/Kana: 待つ : 待ちます = 働く : x
Hepburn: *matsu* : *machimasu* = *hataraku* : x
Kunrei: *matu* : *matimasu* = *hataraku* : x
x = *hatarakimasu*

The algorithm does not solve the first two analogies (solutions: 働きます, *hatarakimasu*) because it does not solve the elementary analogies, つ : ち = く : き and *tsu* : *chi* = *ku* : *ki*, which are beyond the symbol level[7].

More generally speaking, the interaction of analogy with coding seems the basis of a frequent reasoning principle:

$$f(A) : f(B) = f(C) : \mathrm{x} \quad \Leftrightarrow \quad A : B \equiv C : f^{-1}(\mathrm{x})$$

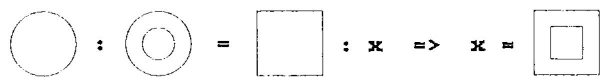
Only the first analogy holds on the symbol level and, as is, is solved by our algorithm. f is an encoding function for which an inverse exists. A striking application of this principle is the resolution of some Copycat puzzles, like:

abc : *abd* = *ijk* : x ⇒ x = *ijl*

Using a binary ASCII representation, which reflects sequence in the alphabet, our algorithm produces:

011000010110001001100011 : 011000010110001001100100
= 011010010110101001101011 : x
⇒ x = 011010010110101001101100 = *ijl*

Set in this way, even analogies of geometrical type can be solved under a convenient representation.



An adequate description (or coding), with no reduplication, is:

$$\begin{array}{l} obj(big)\& \\ obj{=}circle \end{array} : \begin{array}{l} obj(small)\subset obj(big) \\ \& obj{=}circle \end{array} = \begin{array}{l} obj(big)\& \\ obj{=}square \end{array} : \mathrm{x}$$

This is actually solved by our algorithm:

$$\mathrm{x} = \begin{array}{l} obj(small)\subset obj(big) \\ \& obj{=}square \end{array}$$

[7]One could imagine extending the algorithm by parametrising it with such predefined analogical relations.

In other words, coding is the key to many analogies. More generally we follow (Itkonen and Haukioja 97) when they claim that analogy is an operation against which formal representations should also be assessed. But for that, of course, we needed an automatic analogy-solver.

Conclusion

We have proposed an algorithm which solves analogies on words, *i.e.* when possible it coins a fourth word when given three words. It relies on the computation of pseudo-distances between strings. The verification of a constraint, relevant for analogy, limits the computation of matrix cells, and permits early termination in case of failure.

This algorithm has been proved to handle many different cases in many different languages. In particular, it handles parallel infixing, a property necessary for the morphological description of semitic languages. Reduplication is an easy extension.

This algorithm is independent of any language, but not coding-independent: it constitutes a trial at inspecting how much can be achieved using only pure computation on symbols, without any external knowledge. We are inclined to advocate that much in the matter of usual analogies, is a question of symbolic representation, *i.e.* a question of encoding into a form solvable by a purely symbolic algorithm like the one we proposed.

A Examples

The following examples show actual resolution of analogies by the algorithm. They illustrate what the algorithm achieves on real linguistic examples.

A.1 Insertion or deletion of prefixes or suffixes

Latin: *oratorem* : *orator* = *honorem* : x
x = *honor*
French: *répression* : *répressionnaire* = *réaction* : x
x = *réactionnaire*
Malay: *tinggal* : *ketinggalan* = *duduk* : x
x = *kedudukan*
Chinese: 科学 : 科学家 = 政治 : x
x = 政治家

A.2 Exchange of prefixes or suffixes

English: *wolf* : *wolves* = *leaf* : x
x = *leaves*

Malay: *kawan* : *mengawani* = *keliling* : x
x = *mengelilingi*

Malay: *keras* : *mengeraskan* = *kena* : x
x = *mengenakan*

Polish: *wyszedleś* : *wyszlaś* = *poszedleś* : x
x = *poszlaś*

A.3 Infixing and umlaut

Japanese: 乗る : 乗せる = 寄る : x
x = 寄せる

German: *lang* : *längste* = *scharf* : x
x = *schärfste*

German: *fliehen* : *er floh* = *schließen* : x
x = *er schloß*

Polish: *zgubiony* : *zgubieni* = *zmartwiony* : x
x = *zmartwieni*

Akkadian: *ukaššad* : *uktanaššad* = *ušakšad* : x
x = *uštanakšad*

A.4 Parallel infixing

Proto-semitic: *yasriqu* : *sariq* = *yanqimu* : x
x = *naqim*

Arabic: *huzila* : *huzāl* = *ṣudi'a* : x
x = *ṣudā'*

Arabic: *arsala* : *mursilun* = *aslama* : x
x = *muslimun*

References

Robert I. Damper & John E.G. Eastman
Pronouncing Text by Analogy
Proceedings of COLING-96, Copenhagen, August 1996, pp. 268–269.

Dedre Gentner
Structure Mapping: A Theoretical Model for Analogy
Cognitive Science, 1983, vol. 7, no 2, pp. 155–170.

Rogers P. Hall
Computational Approaches to Analogical Reasoning: A Comparative Analysis
Artificial Intelligence, Vol. 39, No. 1, May 1989, pp. 39-120.

Douglas Hofstadter and the Fluid Analogies Research Group
Fluid Concepts and Creative Analogies
Basic Books, New-York, 1994.

Robert R. Hoffman
Monster Analogies
AI Magazine, Fall 1995, vol. 11, pp 11-35.

Esa Itkonen
Iconicity, analogy, and universal grammar
Journal of Pragmatics, 1994, vol. 22, pp. 37-53.

Esa Itkonen and Jussi Haukioja
A rehabilitation of analogy in syntax (and elsewhere)
in András Kertész (ed.) *Metalinguistik im Wandel: die kognitive Wende in Wissenschaftstheorie und Linguistik* Frankfurt a/M, Peter Lang, 1997, pp. 131-177.

Yves Lepage & Ando Shin-Ichi
Saussurian analogy: a theoretical account and its application
Proceedings of COLING-96, Copenhagen, August 1996, pp. 717–722.

Nagao Makoto
A Framework of a Mechanical Translation between Japanese and English by Analogy Principle
in *Artificial & Human Intelligence*, Alick Elithorn and Ranan Banerji eds., Elsevier Science Publishers, NATO 1984.

Vito Pirelli & Stefano Federici
"Derivational" paradigms in morphonology
Proceedings of COLING-94, Kyoto, August 1994, Vol. I, pp 234-240.

Victor Sadler and Ronald Vendelmans
Pilot implementation of a bilingual knowledge bank
Proceedings of COLING-90, Helsinki, 1990, vol 3, pp. 449-451.

Eric Steinhart
Analogical Truth Conditions for Metaphors
Metaphor and Symbolic Activity, 1994, 9(3), pp 161-178.

Esko Ukkonen
Algorithms for Approximate String Matching
Information and Control, 64, 1985, pp. 100-118.

Robert A. Wagner and Michael J. Fischer
The String-to-String Correction Problem
Journal for the Association of Computing Machinery, Vol. 21, No. 1, January 1974, pp. 168-173.

François Yvon
Paradigmatic Cascades: a Linguistically Sound Model of Pronunciation by Analogy
Proceedings of ACL-EACL-97, Madrid, 1994, pp 428-435.

UN ALGORITHME POUR LA RÉSOLUTION DES ANALOGIES ENTRE MOTS

Yves LEPAGE

Résumé

Un rappel de travaux précédents sur l'analogie en psychologie, en intelligence artificielle et en traitement automatique des langues précède la présentation d'un algorithme de résolution, au niveau morphologique, d'analogies entre mots. Cet algorithme crée un quatrième mot à partir de trois mots donnés, quand c'est possible. Par exemple, étant donnés *fable*, *fabuleux* et *miracle*, l'algorithme crée bien *miraculeux*. Des cas bien plus difficiles sont correctement résolus par l'algorithme, en particulier, les cas d'infixation multiple, nécessaires pour rendre compte de la morphologie des langues sémitiques. Nous donnons les caractéristiques de l'algorithme et mentionnons quelques applications possibles.

ALGORYTM DO ROZSTRZYGANIA ANALOGII POMIĘDZY SŁOWAMI

Yves LEPAGE

Streszczenie

Po opisaniu poprzednich prac nad zagadnieniem analogii w ramach psychologii, sztucznej inteligencji oraz lingwistyki komputerowej, pokazujemy algorytm do rozwiązania analogii pomiędzy słowami na poziomie morfologicznym. Algorytm ten tworzy, kiedy jest to możliwe, czwarty termin na podstawie trzech innych terminów. Na przykład, jeżeli podamy *śpiewać*, *śpiewaczka* i *działać*, algorytm słusznie stworzy *działaczka*. Algorytm ten rozwiązuje bardziej skomplikowane problemy analogii, jak w przypadku morfologii języków semityczych, gdzie w środku słów może pojawić się kilka przyrostków jednocześnie. Opisujemy algorytm i jego możliwe zastosowania.

EIN ALGORITHMUS ZUR LÖSUNG VON WORT-ANALOGIEN

Yves LEPAGE

Zusammenfassung

Nach einer Beschreibung früherer Werke über Analogie im Rahmen von Psychologie, künstlicher Intelligenz und maschineller Sprachverarbeitung, wird ein Algorithmus zur Lösung von Wort-Analogien auf morphologischer Ebene vorgeschlagen. Dieser Algorithmus erzeugt, wenn möglich, ein viertes Wort aus drei gegebenen Wörtern. Zum Beispiel, *aussähest* wird aus *nehmen*, *ausnähmest* und *sehen* abgeleitet. Auch komplexere Fälle werden korrekt behandelt, selbst in der Morphologie semitischer Sprachen, in denen parallele Infixung vorkommt. Der Algorithmus wird beschrieben und mögliche Anwendungen werden aufgezeigt.

単語の類推関係解決手法

Yves LEPAGE (ルパージュ)

概要

心理学、人工知能、計算言語学において類推の問題を取り扱った先行研究が、単語の類推を解くアルゴリズムの基礎となっている。単語に関する類推とは、ある三語、例えば「食べる」「食べない」「決める」が与えられたときに第四の単語「決めない」を出力することである。類推はより複雑な関係、例えばセム語族の処理に必要な複数の接中辞の挿入や置換も取り扱うことが可能である。本論文では、アルゴリズムの計算的な特徴を述べると共に、いくつかの適用例について言及する。