MULTIPLE ALIGNMENT, COGNITION AND COMPUTING

J Gerard Wolff

September 1996

Dr J. G. Wolff, School of Electronic Engineering and Computer Systems, University of Wales, Dean Street, Bangor, Gwynedd, LL57 1UT, UK. Telephone: +44 1248 382691. E-mail: gerry@sees.bangor.ac.uk. Fax: +44 1248 361429. Web: http://www.sees.bangor.ac.uk/~gerry.

Abstract

This article is an overview of new proposals with potential to integrate and simplify a range of concepts and observations in computing, cognition and AI.

Examples are presented to suggest how information compression by 'multiple alignment', 'unification' and search (ICMAUS) may serve as a unifying framework in cognitive science. In this context, 'multiple alignment' has a meaning which is close to its meaning in bio-informatics; and 'unification' is used to mean a simple merging of matching patterns, a meaning which is simpler than but related to the meaning of that term in logic.

ICMAUS appears to provide a framework within which the following ideas may be modelled: the concepts of 'variable' and 'types' for variables; the operation of a Turing machine; unsupervised learning of classes from examples; unsupervised learning of a grammar from examples; syllogistic reasoning (*modus ponens*); abductive reasoning; chains of reasoning; the role of inheritance in creating an instance of a class; the execution of simple and composed functions; parsing of a sentence; creation of a sentence; the processing of discontinuous dependencies in syntax; best-match information retrieval with or without indirection; 'fuzzy' pattern recognition; problem solving (jigsaw puzzles and geometric analogy problems).

Keywords

information compression; multiple alignment; unification; search; variable; Turing machine; learning; reasoning; function; parsing; information retrieval; pattern recognition; problem solving.

1. Introduction

This article is a summary or overview of ideas conceived recently within a programme of research [8-20] developing the 'SP' conjecture that:

> *All kinds of computing and formal reasoning may usefully be understood as information compression by pattern matching, unification[1] and search.*

The motivations for developing this conjecture are:

- The possible integration and simplification of diverse concepts in computing, cognition and AI.
- The possibility of developing a 'new generation' computing system with more flexibility and 'intelligence' than conventional computers.

The key insight in this latest phase of the research is that a concept of *multiple alignment* (with 'unification' and 'search') seems to provide a model for 'computing' in some deep sense and for several other concepts associated with computing, cognition and AI. As described below, the term 'multiple alignment' has a meaning which is similar to its meaning in bio-informatics

The intention of this article is to suggest the power of the multiple alignment concept by showing with examples how it may accommodate a wide range of concepts in cognitive science.

Within the space of one article, the treatment of each topic must necessarily be brief. Fuller discussion some of the topics may be found in [8-12].

2. Information Compression by Multiple Alignment, Unification and Search[2]

Figure 1 shows five DNA sequences arranged in such a way that matching symbols are aligned in vertical columns. This kind of 'multiple alignment' of DNA sequences or of sequences of amino acid residues is used in biochemistry as part of the process of elucidating the structure, functions or evolution of the corresponding molecules.

For any given set of sequences, there is normally an astronomically large number

1. The term *unification* is used in this programme of research means a simple merging of multiple instances of any pattern to make one. This idea is related to, but simpler than, the concept of 'unification' as it is used in logic.
2. In the rest of this article, it will be convenient to abbreviate "information compression by multiple alignment, unification and search" as "ICMAUS".

of alternative alignments, some better than others. What may be meant by a ‘good’ alignment is described briefly below.

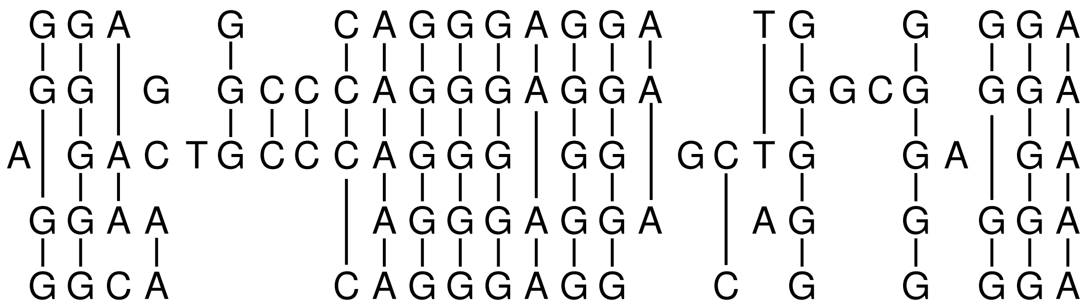


Figure 1. An alignment of five DNA sequences (adapted from Fig. 6 in Ref. 6, with permission from Oxford University Press).

2.1. GENERALISATION OF THE CONCEPT OF ‘MULTIPLE ALIGNMENT’

In the SP programme, the concept of multiple alignment has been generalised to include cases where a given sequence may appear two or more times in different parts of the alignment. In this generalised version of the concept, any given sequence may be aligned with itself, with the obvious restriction that any one symbol cannot be matched with itself.

2.2. COMPRESSION AND UNIFICATION

In the SP programme, and some of the biochemical applications of multiple alignment, a ‘good’ alignment is one which can be compressed into a compact form using some kind of coding scheme [1, 2, 3, 11]. Generally speaking, this means alignments in which there are many ‘hits’ (positive matches between symbols) and relatively few unmatched symbols between the hits.

In the SP programme, there is a distinction between ‘New’ information (which may be regarded as ‘input’ to a ‘program’ or as a ‘query’ for an expert system or database) and ‘Old’ information (which is analogous to a ‘program’, to ‘rules’ in an expert system or to information stored in a ‘database’). A ‘good’ alignment is one which allows New information to be encoded economically in terms of Old information.

As will be seen (Section 3.2), the key to compression is the merging or ‘unification’ of matching sequences or subsequences of symbols, with or without the insertion of ‘tags’ to mark where information has been deleted.

2.3. COMPRESSION AND PROBABILITIES

Evaluation of an alignment in terms of compression can, in principle, be translated into an equivalent evaluation in terms of probability. Compression and probability are two sides of the same coin [1, 2, 3, 11].

2.4. METRICS-GUIDED SEARCH

Finding good alignments means searching the abstract 'space' of possible alignments. Since this space is normally too large to be searched exhaustively, it is normally necessary to use some kind of 'heuristic' technique ('hill climbing', 'simulated annealing', 'genetic algorithm' etc), sometimes known generically as 'metrics-guided search'. These kinds of search can deliver acceptably good results for a reasonable computational effort but, for any realistically large body of information, when all parts of the search space can potentially be entered, they cannot guarantee that the best possible result will be found.

Of course, it is always possible to constrain this kind of search so that selected parts of the search space can never be reached. If the search space is narrowed in this way, it may become possible to guarantee that the best possible results will always be found. In cases like this, it seems that the kind of predictability found in conventional systems may be achieved but with offsetting losses in flexibility and 'intelligence'.

3. Basic Ideas

This section outlines some ideas which are basic in this developing framework and which recur in the examples which follow.

3.1. REPRESENTATION OF KNOWLEDGE

In the SP framework, it is assumed that *all* kinds of knowledge may be expressed as 'patterns' of atomic symbols, where a pattern is an array of symbols in one, two or more dimensions. For the time being in this research programme, attention is confined to one-dimensional patterns.

In this framework, there are no 'meta' symbols with special meaning. All symbols enter into matching and unification in the same way. A working hypothesis is that *all* the representational devices used in other notations may be derived from patterns of atomic symbols in an ICMAUS framework

3.2. COMPRESSION AND THE USE OF 'TAGS'

Unification of the two instances of 'A B C D E F G H I J K' in the two sequences

'1 2 A B C D E F G H I J K 3 4 5' and '6 7 8 A B C D E F G H I J K 9'allows those two sequences to be reduced to:

1 2 x ; 3 4 5
6 7 8 x ; 9
x **A B C D E F G H I J K** ;

In this and later examples, bold type is used to mark symbols which are the result of the unification of two or more matching symbols.

The pattern 'x ;' has been introduced to mark the positions of 'A B C D E F G H I J K' in the two contexts in which it originally appeared. The symbols 'x' and ';' have also been attached at the beginning and end of 'A B C D E F G H I J K'.

This idea of deleting repeated instances of a pattern and replacing each instance with a relatively short 'code' or 'tag' is the basis of most of the simpler 'standard' techniques for information compression.

Variations on this idea are discussed in [11]. Also discussed there is the reason for introducing a 'termination marker' like the symbol ';' in the example.

3.3. DE-COMPRESSION AND THE 'DE-REFERENCING' OF TAGS

With the example just shown, either of the original sequences may be recreated by 'de-referencing' the tag pattern in the reduced version of the given sequence. This may be achieved by alignment and unification of patterns, as shown in Figure 2.

```
Alignment:      1 2 x                       ; 3 4 5
                    |                       |
                    x A B C D E F G H I J K ;

Unification:    1 2 x A B C D E F G H I J K ; 3 4 5
```

Figure 2. 'De-referencing' of a tag pattern by alignment and unification of patterns.

As can be seen from this example, the tag symbols enter into matching and unification in exactly the same way as other symbols. Apart from the fact that they may have been added to stored knowledge by the system, they have no special status. They are not 'meta' symbols and have no hidden meaning.

3.4. IMITATING THE EFFECT OF A 'VARIABLE' WITH A 'TYPE'

In the example, the pattern 'x ;' behaves like a named 'variable' in a conventional computing system: it is a place-marker for missing information which, in the example, may take 'A B C D E F G H I J K' as its 'value'.

In other cases, there may be a set of alternative values for such a 'variable'. For example, with these three patterns:

A B x ; C D E
x 0 ;
x 1 ;

the 'variable' 'x ;' may take '0' or '1' as its 'value'. By defining the range of values which the 'variable' may take, the two patterns 'x 0 ;' and 'x 1 ;' serve as a 'type definition' for the 'variable'. In this case, the Boolean type has been defined extensionally in a manner comparable with the definition of an 'enumerated type' in a language like Pascal.

3.4.1. Types with Infinite Range

Of course, many variables in conventional programming systems have type definitions like 'integer' or 'alphabetic string' which represent an infinite range of possible values.

As an example of how typing with an infinite range may be modelled within the ICMAUS framework, the type 'binary natural number' may be defined with these three patterns:

x ;
x 0 x
x 1 x.

Given these three patterns, the 'variable' in a pattern like 'A B x ; C D' may receive any binary natural number as its 'value' by alignment and unification of relevant patterns, as shown in Figure 3.

Notice that two of the patterns ('x 0 x' and 'x 1 x') each appear more than once in the alignment in accordance with the generalisation of the alignment problem described in Section 2.1.

In this example, the three patterns which define the 'type' of the 'variable' behave like simple re-write rules:

x →
x → 0 x
x → 1 x.

The first of these rules expresses the idea that any given binary natural number may be null. The second and third rules define (recursively) any sequence of ‘0’ and ‘1’.

```
Alignment:          1   0   1   1   1   0
                    |   |   |   |   |   |
              A B x |   |   |   |   |   |          ; C D E
                  | |   |   |   |   |   |          |
                  x 1 x |   |   |   |   |          |
                      | |   |   |   |   |          |
                      x 0 x |   |   |   |          |
                          | |   |   |   |          |
                          x 1 x |   |   |          |
                              | |   |   |          |
                              x 1 x |   |          |
                                  | |   |          |
                                  x 1 x |          |
                                      | |          |
                                      x 0 x        |
                                          |        |
                                          x        ;

Unification:  A B x 1 x 0 x 1 x 1 x 1 x 0 x ; C D E
```

Figure 3. Alignment and unification of patterns showing how a ‘variable’ (‘x ;’) may receive a binary natural number (‘1 0 1 1 1 0’) as its ‘value’.

3.4.2. Separation of Type Definitions from Variable Names

By contrast with the examples just shown, conventional systems normally define types independently of any specific variable and then associate types with variables by putting their respective identifiers next to each other when each variable is declared (e.g., ‘int i’). It appears that this convention may be modelled in the ICMAUS framework but space does not permit an explanation here.

4. The Operation of a Turing Machine

This section and the ones which follow present simple examples intended to suggest briefly but clearly how a range of other concepts in computing and AI may be modelled as

ICMAUS.

The first example - how the operation of a Turing machine may be modelled in this framework - has a bearing on the generality of the new proposals because a Turing machine is widely regarded as a general model of computing. If it can be shown that the operation of any Turing machine can be modelled as ICMAUS, then the latter may also be regarded as a general model of computing. Since it is generally accepted that the operation of any Turing machine may be modelled with a 'Post canonical system' (PCS) (see [5] for useful discussion), then it will be sufficient to show that the operation of a PCS may be modelled with ICMAUS.

In its simplest 'normal' form, a PCS comprises an 'alphabet' of symbols, one 'primitive assertion' (which is a string of symbols in the alphabet) and zero or more 'productions', each one with the form:

$g\$ \rightarrow \$h$

where g and h each represent a string of zero or more symbols, and both instances of '$' represent a single 'variable' which may have a 'value' comprising a string of zero or more symbols. Any kind of PCS may be modelled by a PCS in normal form.

To see how the system works, consider a PCS with an alphabet comprising the symbols 0 to 9, the primitive assertion '4 1 5 9 3' (which may be regarded as 'input') and these two productions:

4 1 $ → $ 7 1 8
5 9 3 7 $ → $ 2 6 5.

The system seeks a match between the initial symbols in the input string and the left-hand side of any of the productions, with the variable in a successfully matched production accepting as its value any trailing symbols in the input string which have not been matched. In the example, the first two symbols in the input string match the first two symbols in the first production and the variable receives '5 9 3' as its value. Since '$' on the left-hand side of the production represents the same value as '$' on the right-hand side of the production, the result of this operation is the creation of a new string '5 9 3 7 1 8'.

This newly-created string is treated as if it were new input. In this case, the initial symbols in the string match the symbols on the left-hand side of the second production, the variable receives '1 8' as its value and the result is the new string '1 8 2 6 5'.

This process continues as long as matches can be found. In this example, the new string does not match the left-hand side of any of the productions so it may be regarded as the final result of the computation.

In keeping with the ideas described in Section 3, the two productions in this example (with a 'type definition' of each 'variable') may be recast as:

4 1 $; 7 1 8

5 9 3 7 $; 2 6 5

$;

$ 0 $

...

$ 9 $

In each of the first two patterns, the symbols '$;' represent the 'variable' in the corresponding production. As will be seen, it does not seem necessary to have two instances of the variable symbol in each production and the arrow ('→') does not seem to be needed either.

The processing of the string '4 1 5 9 3' as described above may be seen as alignment and unification of the input pattern with patterns representing the productions and possible values of each variable. What appears to be the best alignment of relevant patterns is shown in Figure 4.

The corresponding unification:

4 1 **$ 5 $ 9 $ 3 $; 7 $ 1 $ 8 $;** 2 6 5

contains, as subsequences, both the intermediate string ('5 9 3 7 1 8') and the final result ('1 8 2 6 5'). Thus multiple alignment with unification and search may be seen to model the operation of the corresponding PCS and its equivalent Turing machine.

The key difference between the Turing/PCS models and ICMAUS is in the search processes embodied in the models. Formal descriptions of Turing and PCS models seldom include a formal description of the search process. However, it is apparent that these models each require a search process and that relatively simple and constrained search processes are assumed. By contrast, the SP/ICMAUS proposals exploit a concept of search which has been generalised explicitly into the realms of partial matching and multiple alignment. This generalisation of the search concept embodied in the model appears to be the key to achieving more intrinsic flexibility and 'intelligence' in computing systems.

The ICMAUS proposal also differs from the PCS (and other systems with a similar character, e.g., [7]) because there is no need for 'meta' symbols like '$', '→' and so on. The ICMAUS framework may be applied to any (sequential) information, including 'raw' information which, by its nature, would not normally contain meta symbols

A fuller description of these ideas is presented in [8], including more detail of the way recursive operations may be accommodated in the framework.

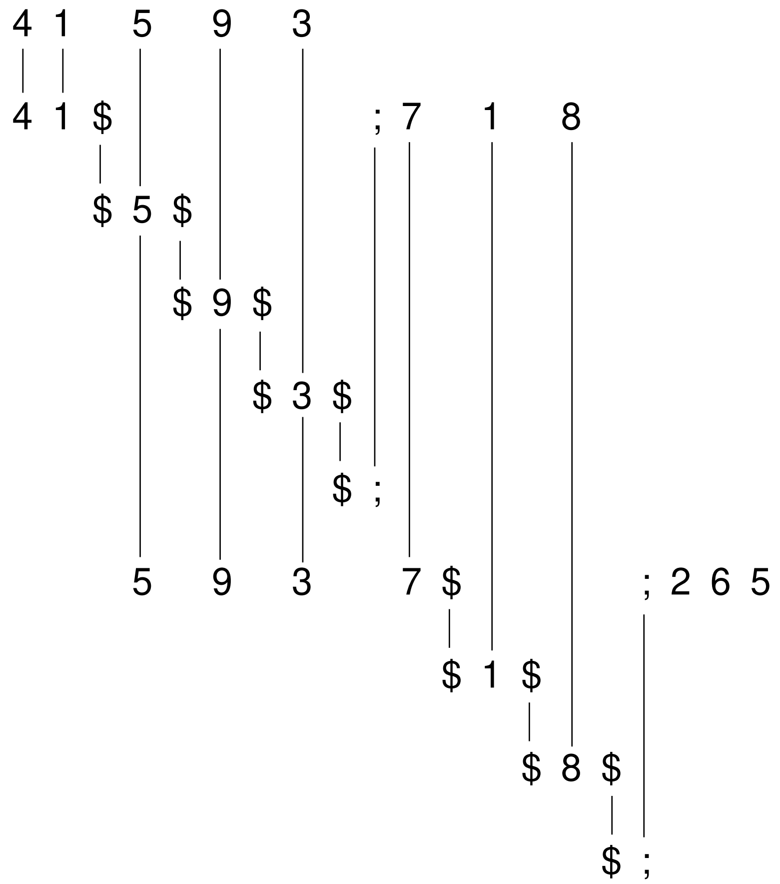


Figure 4. An alignment of patterns modelling the operation of a PCS.

5. Learning

5.1. UNSUPERVISED LEARNING OF CLASSES FROM EXAMPLES

A class may be abstracted from a set of examples by alignment and unification of patterns as shown in Figure 5 (see also [14], Section 5.4). In the figure, the features of three different canal boats for hire in the UK are matched and unified to form a generalised description of a canal boat plus residual patterns showing the distinguishing features of each individual boat. Tags are inserted so that the original patterns can be re-created by alignment and unification as described in Section 6.4

Readers may wonder why the matching 'o' letters in 'Avon' and 'Royal' and the matching 'n' letters in 'Avon' and 'Trent' have not been coded. The reason is that no net compression is achieved by encoding matches as small as these, so unification and tagging is not justified.

It should be clear that this idea of merging the matching parts of patterns and abstracting the parts which do no match may be generalised to the learning of class hierarchies with any number of levels.

Alignment:

```
nameAvo  n    length56'berths 6kitchenbathroom
|||||    |    ||||||| |||||||  |||||||||||||||
nameR  oy|al length69'berths10kitchenbathroom
||||     |    ||||||| |||||||  |||||||||||||||
nameTre  nt   length62'berths 8kitchenbathroom
```

Unification:

name N; **length** L; **'berths** B; **kitchenbathroom**
NAvon;L56;B6;
NRoyal;L69;B10;
NTrent;L62;B8;

Figure 5. How a class may be abstracted from a set of examples by alignment and unification of patterns. The examples are canal boats for hire.

This idea may also be generalised to the learning of class heterarchies in which a given instance or class may appear in more than one super-class (ie, 'cross-classification' or 'multiple inheritance').

Of course, 'natural' classes are a little more subtle than this. In particular, they often exhibit the property of 'polythesis': no single attribute need be present in every member of the class. It would take too much space to discuss these matters fully but it seems that polythesis in natural categories may be modelled by 'grammars' in which there are 'disjunctive' categories (see [21]). The way such grammars may be learned in an ICMAUS framework is described briefly in Section 5.2.

5.2. UNSUPERVISED LEARNING OF A GRAMMAR FROM EXAMPLES

Figure 6 shows a very simple example of how a grammar may be learned by alignment and unification of example sentences.

Alignment:

```
J a n e                   r u n s
| | | |                   | | | |
| | | | P e t e r w a l k | | | s
| | | | | | | | | | | | | | | | |
J a n e | | | | | w a l k | | | s
        | | | | |         | | | |
        P e t e r         r u n s
```

Unification:

N **J a n e** ;

N **P e t e r** ;

V **r u n** ;

V **w a l k** ;

N ; V ; **s**

Figure 6. How a simple grammar may be learned by multiple alignment and unification.

As will be seen in Section 8, patterns like those which are the result of unification in the example, may function as re-write rules in a grammar.

In more complex cases, patterns like ‘N ; V ; s’ which are produced by unification at one stage may themselves be the subject of further alignment and unification at later stages. In this way, arbitrarily deep hierarchical grammars may be built up.

The way in which this kind of alignment and unification of patterns may enable the system to learn ‘discontinuous dependencies’ in syntax is described and discussed in [14], Section 5.3.

Of course, there is a lot more to be said about unsupervised grammar learning than this. Important issues include the formation of generalisations, the unsupervised correction of overgeneralisations, learning of syntactic features with sub-categorization of syntactic categories, and the learning of semantics. These matters are considered in [21] and [22].

6. Reasoning

A working hypothesis in this programme of research is that all forms of reasoning may be modelled in a probabilistic framework and that the relative certainties of 'deduction' may be modelled by probability values which are close to 0 or 1.

The way in which negation may be modelled in this framework has not yet received close attention (but see the example in Section 7.2).

6.1. SYLLOGISTIC REASONING

Here is an example of simple one-step reasoning:

All triangles are polygons.
F6 is a triangle.
Therefore, F6 is a polygon.

In logical notation, this may be represented as:

$\forall x$: T(x) $\Rightarrow$ P(x).
T(F6).
$\therefore$ P(F6).

In accordance with the ideas outlined in Section 3, the two premises may be represented with these two patterns:

T x ; P x ;
T F 6.

The first pattern expresses the association between being a triangle and being a polygon. Why is there nothing corresponding to the arrow symbol for implication ('$\Rightarrow$') in the logical notation? Informally, that symbol expresses the idea that there is an asymmetry in the implication: it works well from left to right but not so well in the other direction.[3] Since this asymmetry can, apparently, be accommodated in other ways (see Section 6.2, below), the arrow has been omitted.

The second pattern expresses the association between triangularity and F6.

As before (Section 3), other patterns are needed alongside the two premises to define the (infinite) range of possible values of the 'variable' 'x ;'.

Given these patterns, the inference may be modelled by alignment as shown in Figure 7.

---

3. In a stricter 'logical' interpretation, the symbol expresses 'material implication' which includes a concept of negation. How this more formal interpretation may be accommodated within the ICMAUS framework is considered elsewhere [10].

```
T x ; P x         ;
        |         |
      T x         ; P x ;
      | |         |
      T | F   6   |
        | |   |   |
        x F x |   |
            | |   |
            x 6 x |
                | |
                x ;
```

Figure 7. Alignment of patterns in an example of syllogistic reasoning.

The first instance of the 'variable' 'x ;' in the pattern 'T x ; P x ;' receives 'F 6' as its 'value'. Since the two instances of 'x ;' are aligned, the value is, in effect, 'transferred' from the first instance to the second so that we may reach the conclusion that 'P' and 'F 6' are associated, meaning that "F6 is a polygon".

6.2. ABDUCTION

It should be apparent from the example just given that, if the second premise were that "F6 is a polygon", the ICMAUS framework would lead one to the (probabilistic) conclusion that F6 could be a triangle. Figure 8 shows a corresponding alignment.

Despite the risks, this is a valid form of reasoning which people use constantly. The probability attaching to this inference would depend on the other patterns in the database from which alternative, competing, alignments might be formed. If the database records the facts that squares, pentagons, hexagons etc. are polygons, then a relatively low probability would attach to the inference that F6 was a triangle. Conversely, if these other associations had not been recorded, the system, in its ignorance, would infer that the terms 'polygon' and 'triangle' were equivalent and that if F6 is a polygon it must necessarily also be a triangle.

In general, it seems that the asymmetry which may exist amongst inferences which, traditionally, is expressed with symbols like '⇒', may be accommodated by differences in probability between 'left' and 'right' alignments which themselves depend on the assembly of patterns which the system knows.

```
          T x             ; P x ;
            |             |
      T x ; P x           ;
          | |             |
          P | F   6       |
            | |   |       |
            x F x |       |
                | |       |
                x 6 x     |
                    |     |
                    x ;
```

Figure 8. Alignment of patterns in an abductive inference.

6.3. CHAINS OF REASONING

The concept of multiple alignment provides a framework in which ‘chains’ of reasoning and more subtle combinations of reasons seem to fit quite comfortably. Figure 9 shows how the kinds of inferences which may be made in a case of suspected arson may be cast as an alignment of relevant patterns.

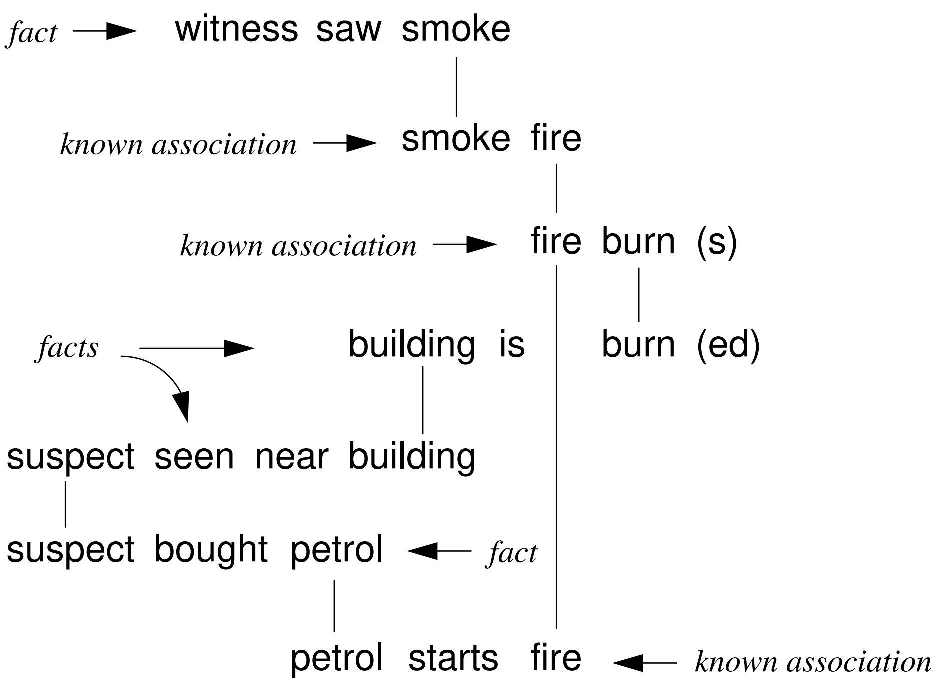


Figure 9. Alignment of patterns showing inferential connections in a case of suspected arson. Words in italics with arrows are merely commentary and are not part of the alignment.

In this case, no one actually saw the fire but the existence of a fire is inferred via well-known associations from the fact that a witness saw smoke and that a building has been burned. No one actually saw the suspect start the fire but this may be inferred from the facts that he is known to have bought petrol and was seen near the building, and the general knowledge that petrol is a means of starting a fire. If the suspect had also recently taken out insurance on the building this would strengthen the case via other well-known associations and motivations.

6.4. CLASSES AND INHERITANCE

A form of reasoning which is often implicit in discussion and debate, is the 'inheritance' of attributes of a class or classes by individual members of those classes or by lower-level classes. Knowing that 'John' is a 'person' means that knowledge of the properties of people can be brought to bear on any argument about John and his behaviour.

The way in which this kind of inheritance may be modelled with ICMAUS is shown in Figure 10, using the example from Section 5.1.

Alignment:

```
nameN       ;lengthL   ;'berthsB  ;kitchenbathroom
    |       |        | |        | |
    NAvon;           L56;       B6;
```

Unification:

name**N**Avon**;**length**L**56**;**'berths**B**6**;**kitchenbathroom

Figure 10. Inheritance of descriptors by an instance of a class.

In effect, the instance ‘Avon’ inherits the descriptors ‘name’, ‘length’, ‘berths’, ‘kitchen’ and ‘bathroom’ from the class description with which it unifies.

It should be clear that the framework may accommodate inheritance through any number of ‘levels’.

As described in [9], the ICMAUS framework may also model ‘cross-classification’ or ‘multiple inheritance’.

7. The Execution of Functions

The very general concept of a ‘function’ may be defined as a table of input values, each one with a corresponding output value. With many functions, the range of input and output values in this kind of extensional representation can be very large (often infinite) and it is more convenient to define such a function intensionally, in a compressed form, as a ‘composition’ of simpler functions.

The way in which the design and execution of functions may be understood as information compression is described and discussed in [16] with examples from the SP6 model.

7.1. THE EXECUTION OF SIMPLE FUNCTIONS

Any simple function may be represented and used as a table of input values with corresponding output values. Such a table may be represented with a set of one-dimensional patterns, one for each row, as shown here for the ‘equivalence’ function:

1 1 ; 1 ;
1 0 ; 0 ;
0 1 ; 0 ;
0 0 ; 1 ;

As shown in Figure 11, such a function may be ‘executed’ by alignment and unification between an incomplete pattern representing ‘input’ and patterns in the definition of the function.

Alignment:

```
1 0 ;   ;   <-- 'Input' pattern
| | |   |
1 0 ; 0 ;   <-- A pattern in the function definition
```

Unification:

1 0 ; 0 **;**

Figure 11. Execution of the equivalence function by alignment and unification of patterns.

The alignment shown in the figure is the best of several possible alignments between the ‘input’ pattern ‘1 0 ; ;’ and patterns in the definition. In the corresponding unification, the ‘output’ of the computation is the unmatched symbol ‘0’, which is, of course, the ‘correct’ result.

7.2. THE EXECUTION OF COMPOSED FUNCTIONS

In conventional systems, any function may be ‘composed’ (recursively) from simpler functions. A similar effect may be achieved in a Prolog program where the body of a Horn clause may reference two or more ‘procedures’ specified elsewhere in the program. For example, a Prolog procedure which calculates the negation of the equivalence function may be defined thus:

```
not_equiv(X, Y, R) :- equiv(X, Y, T), not(T, R).

equiv(1, 1, 1).
equiv(1, 0, 0).
equiv(0, 1, 0).
equiv(0, 0, 1).

not(1, 0).
not(0, 1).
```

The key point of interest here is the way the ‘T’ variable in the first line acts as a channel through which the ‘output’ of the ‘equiv’ procedure can become the ‘input’ of the ‘not’ procedure. This is achieved because of the convention in Prolog that two or more instances of a variable name in one Horn clause represent a single variable. Any value acquired by the variable in one location is automatically the same in all other locations in the same clause.

This kind of effect may be imitated with ICMAUS. The following patterns may be seen as a version of the above Prolog program, simplified by removing the initial ‘X’, ‘Y’ and ‘R’ but retaining the two instances of ‘T’:

```
not_equiv equiv X ; Y ; T ; % not T ; R ; % #

equiv 1 1 1 %
equiv 1 0 0 %
equiv 0 1 0 %
equiv 0 0 1 %

not 1 0 %
not 0 1 %

X 1 ;
X 0 ;
Y 1 ;
Y 0 ;
T 1 ;
T 0 ;
R 1 ;
R 0 ;
```

The last eight patterns may be seen as 'type definitions' of the 'variables', as described in Section 3.

Figure 12 shows what appears to be a good alignment of a 'call' pattern, 'not_equiv 1 0 R ; #', with patterns in the above 'program'. Notice that the 'variable' 'R ;' in the call pattern is aligned with 'R ;' in 'not_equiv equiv X ; Y ; T ; % not T ; R ; % #' and that '1' from 'R 1 ;' and 'not 0 1 %' lies between 'R' and ';'. In this way, '1' is 'returned' as the (correct) result.

In this alignment, the two instances of the 'variable' 'T ;' in 'not_equiv equiv X ; Y ; T ; % not T ; R ; % #' are aligned with each other and with '0' in 'T 0 ;', '0' at the end of 'equiv 1 0 0 %', and '0' at the beginning of 'not 0 1 %'. Thus, in a manner comparable with the workings of Prolog, the two instances of 'T ;' in the first line of the definition represent a single value which is simultaneously the 'output' of 'equiv' and the 'input' of 'not'.

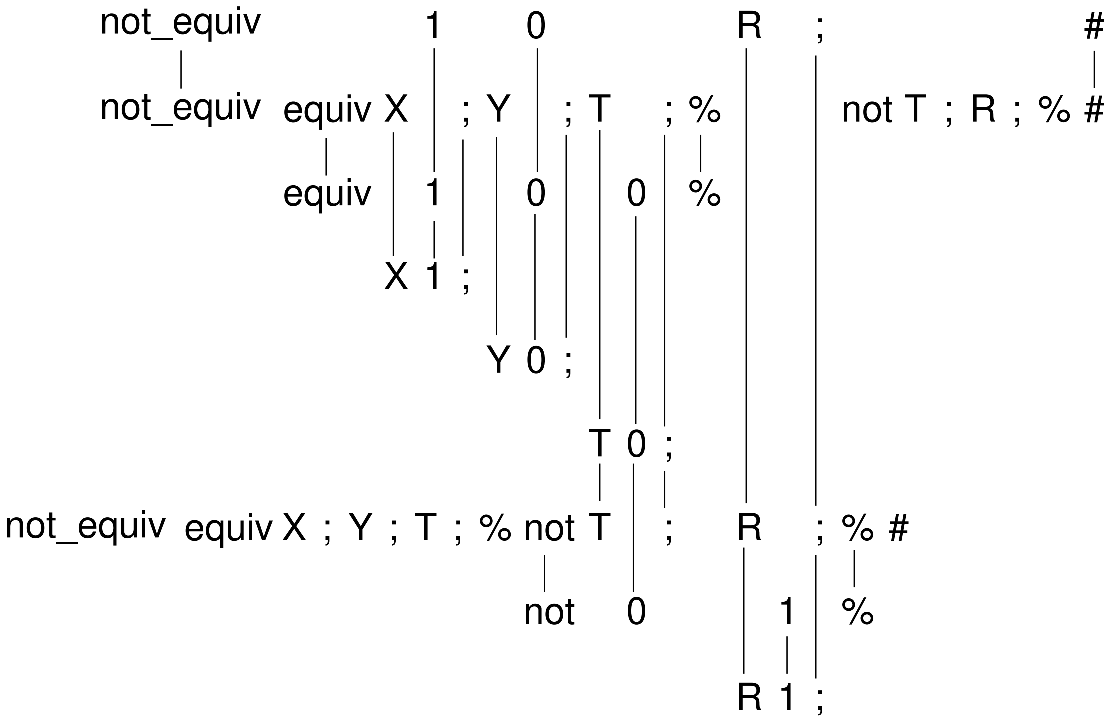


Figure 12. Alignment in the execution of a composite 'function'.

8. Modelling the Use of Grammars

Grammars for simple languages may often be expressed effectively with re-write rules like these:

S → NP VP
NP → J o h n
NP → M a r y
VP → V I ADV
VP → V T NP
V I → r u n s
V I → t a l k s
V T → l i k e s
V T → a m u s e s
ADV → f a s t
ADV → s l o w l y

As indicated in Section 3, these kinds of re-write rules may be recast as SP 'patterns'. With this example, the grammar may be represented like this:

```
S NP ; VP% #
NP 0 J o h n ;
NP 1 M a r y ;
VP V I ; Adv ; %
VP V T ; NP ; %
V I 0 r u n s ;
V I 1 t a l k s ;
V T 0 l i k e s ;
V T 1 a m u s e s ;
ADV 0 f a s t ;
ADV 0 s l o w l y ;
```

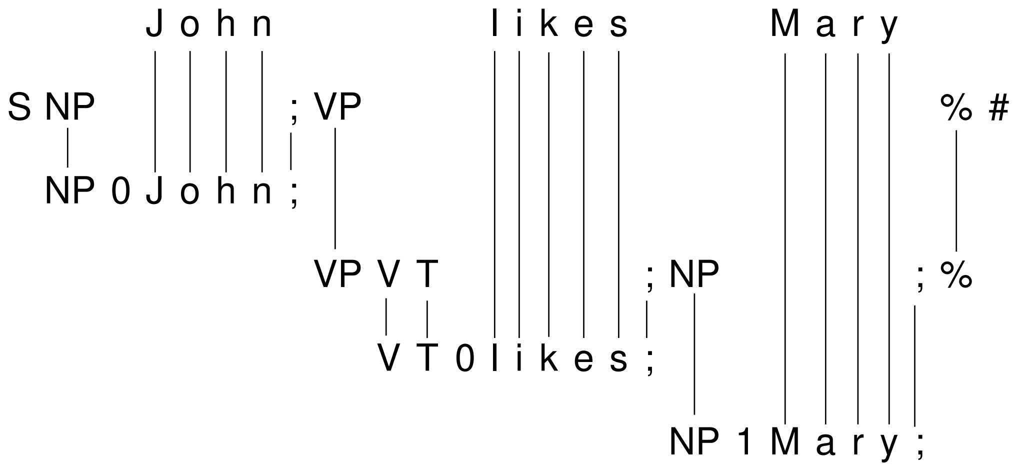
The reason for including the ‘0’ and ‘1’ symbols will be seen in Section 8.2.

With the grammar in this form, the parsing or creation of a sentence may both be seen as ICMAUS, as shown in the next two sections.

8.1. PARSING OF A SENTENCE

Figure 13 shows how the parsing of a simple sentence may be seen in terms of the alignment and unification of the sentence with patterns from the grammar.

Alignment:



Unification:

S **NP** 0 J o h n **; VP V T** 0 l i k e s **; NP** 1 M a r y **; %** #

Figure 13. The parsing of a simple sentence as an alignment and unification of the sentence with grammatical patterns.

It should be apparent that the unification in Figure 13 is, in effect, a labelled bracketing of the sentence, essentially the same as the more conventional:

S(NP(J o h n), VP(V T l i k e s), NP(M a r y))).

8.2. CREATION OF A SENTENCE

Given the grammar shown above, the sentence ‘J o h n l i k e s M a r y’ may be ‘coded’ succinctly as ‘S 0 0 1 #’. Figure 14 shows how, by alignment and unification, the full sentence may be created from the encoding.

At a global level within the grammar, each of the symbols ‘0’ and ‘1’ is ambiguous. But within the *context* provided by the sequence of symbols in ‘S 0 0 1 #’, each instance of those symbols ‘selects’ one of the two nouns or one of the two verbs as appropriate for the corresponding part of the sentence.

.

Alignment:

```
S   0                   0               1             #
S NP                ; VP                             % #
  NP 0 J o h n ;
                   VP V T               ; NP        ; %
                      V T 0 l i k e s ;
                                          NP 1 M a r y ;
```

Unification:

S NP 0 J o h n **; VP V T 0** l i k e s **; NP 1** M a r y **; % #**

Figure 14. Creation of a sentence by alignment and unification of patterns.

8.3. ‘CONTEXT-SENSITIVE’ POWER

One of the nice things about the ICMAUS framework is that it appears to provide a relatively direct and ‘perspicuous’ way of handling the phenomenon of ‘discontinous dependencies’ in syntax.

In a sentence like *The winds from the West are strong*, there is a ‘number’ dependency between the subject (*The winds*) and the verb (*are*). If, as in this case, the subject is plural then the verb must be plural; if the subject is singular the verb must be singular. These dependencies are ‘discontinuous’ because there may be arbitrarily large amounts of structure between the subject and the verb. Dependencies of this kind may overlap, as is the case, for example, with number dependencies and gender dependencies in French.

An example showing how overlapping dependencies may be handled with ICMAUS is presented in [9].

9. Information Retrieval

Information retrieval seems to be largely a matter of finding a match between patterns in a ‘query’ and patterns stored in a database.

In ‘direct’ information retrieval, patterns which are retrieved are similar to patterns in the query in the sense that components of the query match components of the retrieved information. In ‘indirect’ information retrieval, the query and the retrieved information may be similar only in their ‘meanings’.

It seems that both kinds of retrieval may be modelled with ICMAUS.

9.1. DIRECT INFORMATION RETRIEVAL

A simple but effective means of retrieving information from a textual database is to search for one or more ‘good’ matches between a query word or phrase and sequences of letters in the database (see, for example, [15]). Figure 15 shows this kind of match as an alignment between patterns.

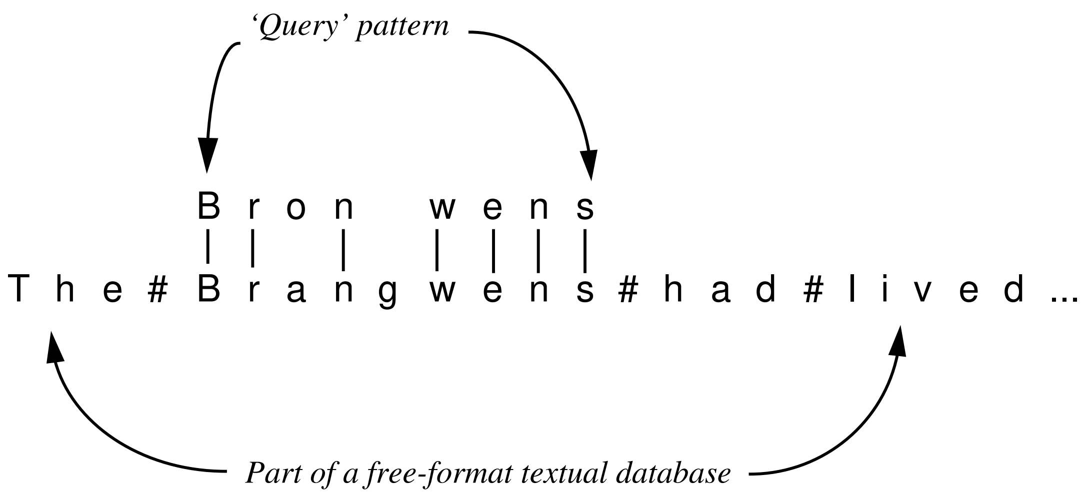


Figure 15. Multiple alignment in best-match information retrieval. The ‘#’ symbol represents space characters in the database.

9.2. INDIRECT INFORMATION RETRIEVAL

A convenience provided by some systems for information retrieval is an ability to search for material which has the same or similar meaning as the query terms as well as material which matches the terms directly. This may be achieved by incorporating a dictionary or thesaurus in the system and using it to extend any set of query terms with synonyms of those terms or with terms which are close in meaning.

This kind of indirection in information retrieval may be seen in terms of the alignment of query patterns with patterns in the dictionary or thesaurus, together with the alignment of the patterns representing meanings with patterns in the database.

10. Pattern Recognition

Pattern recognition is similar to information retrieval because it means finding a 'good' match between some external pattern in one, two or more dimensions and patterns stored in memory. A visual scene, for example, as it is projected onto the retina of an eye or equivalent surface in a video camera, must be 'parsed' in terms of stored patterns. Very often, there is partial occlusion of one object by another. As described in [9], the problem of finding a 'good' match between stored patterns and an external pattern may be seen as a multiple alignment problem, very much like the other examples which have been considered.

11. Problem Solving

This section includes two examples intended to suggest how ICMAUS might serve as a general framework for understanding 'problem solving' in some sense. Clearly, a good deal more work is needed to see whether or not other kinds of problems may be seen in the same terms.

11.1. JIGSAW PUZZLES

The solution to a jigsaw puzzle may be seen as a multiple alignment problem in two dimensions:

- Components of the picture on the pieces of the jigsaw must align with matching components in the 'target' picture.
- Straight edges on the 'edge' pieces of the jigsaw must align with the matching edges of the picture.
- The shape of each piece must match and be aligned with the complementary shapes of neighbouring pieces.

When a jigsaw puzzle is completed, all these matching components are, in effect, 'unified'.

11.2. GEOMETRIC ANALOGIES

Figure 16 shows an example of the kind of geometric analogy problem sometimes used as a puzzle or part of an IQ test.

In order to apply the concept of multiple alignment to such a problem, it is necessary to translate the geometric forms into sequential descriptions like 'small square inside large ellipse' or 'large circle above small triangle'. How this may be done is described in [4] and is not part of the present proposals.

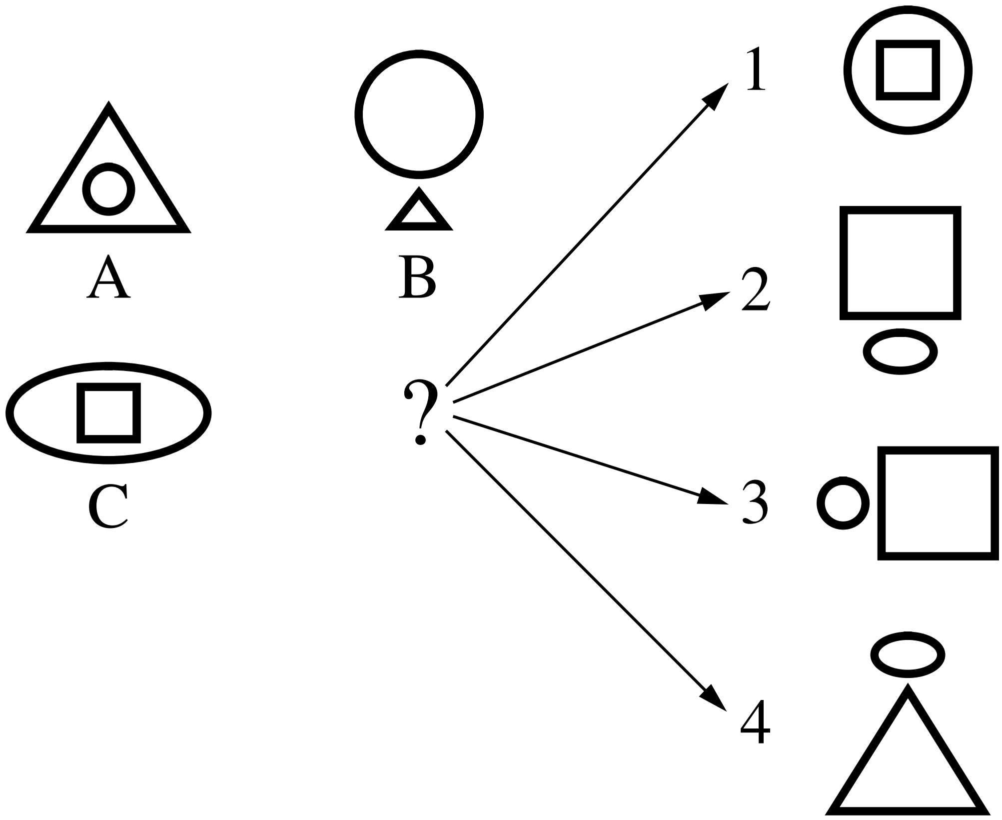


Figure 16. A geometric analogy problem.

For this problem, it appears that item 2 is the ‘correct’ answer. Figure 17 shows this solution to the problem as an alignment of the patterns in the problem after translation into sequential form. It seems that the sequential description of item 2 is the one which fits the other patterns best.

2	large		square	above			small	ellipse
C		small	square		inside	large		ellipse
B	large		circle	above			small	triangle
A		small	circle		inside	large		triangle

Figure 17. A solution to the geometric alignment problem in Figure 16 as an alignment of sequential patterns.

12. Conclusion

As was indicated at the beginning, the intention of this article is to suggest the potential power of the multiple alignment concept by presenting a range of concepts and phenomena in computing and AI to which it may be applied.

Although the treatment of each topic has necessarily been brief, I hope that enough detail has been given to make clear how information compression by multiple alignment, unification and search has potential to illuminate the nature of computing and to integrate and simplify diverse concepts in the field.

References

1 Allison, L., Wallace, C.S. and Yee, C.N. (1992), ‘Finite-state models in the alignment of macromolecules’, *Journal of Molecular Evolution* 35, pp. 77-89.

2 Allison, L. and Yee, C.N. (1990), ‘Minimum message length encoding and the comparison of macromolecules’, *Bulletin of Mathematical Biology* 52 (3), pp. 431-453.

3 Chan, S.C., Wong, A.K.C. and Chiu, D.K.Y. (1992), ‘A survey of multiple sequence comparison methods’, *Bulletin of Mathematical Biology* 54 (4), pp. 563-598.

4 Evans, T.G. (1968), ‘A program for the solution of a class of geometric-analogy intelligence-test questions’, in M. Minsky, ed., *Semantic Information Processing*, Cambridge Mass.: MIT Press, pp. 271-353.

5 Minsky, M.L. (1967), *Computation, Finite and Infinite Machines*. Englewood Cliffs, New Jersey: Prentice Hall.

6 Roytberg, M.A. (1992), 'A search for common patterns in many sequences', *Cabios* 8 (1), pp. 57-64.

7 Strachey, C. (1965), 'A general purpose macrogenerator', *Computer Journal* 7, pp. 225-241.

8 Wolff, J.G. (in preparation), 'Information compression by multiple alignment, unification and search as a general theory of computing'.

9 Wolff, J.G. (in preparation), 'Computing as compression by multiple alignment, unification and search (2)'.

10 Wolff, J.G. (submitted), 'Logic as information compression by multiple alignment, unification and search'

11 Wolff, J.G. (submitted), 'Parsing as information compression by multiple alignment, unification and search'.

12 Wolff, J.G. (1996), 'Learning and reasoning as information compression by multiple alignment, unification and search', in A Gammerman, ed., *Computational Learning and Probabilistic Reasoning*, Chichester, England: Wiley, pp. 67-100. This is a revised version of 'Learning and reasoning as compression by multiple alignment and unification' presented at the Seminar on Applied Decision Technologies '95 (ADT '95), Brunel University, London, April 1995, Stream 1 (Computational Learning and Probabilistic Reasoning), pp. 223-236.

13 Wolff, J.G. (1995), 'Computing as compression: an overview of the SP theory and system', *New Generation Computing* 13, pp. 187-214.

14 Wolff, J.G. (1995), 'Computing as compression: SP20', *New Generation Computing* 13, pp. 215-241.

15 Wolff, J.G. (1994), 'A scaleable technique for best-match retrieval of sequential information using metrics-guided search', *Journal of Information Science* 20 (1), pp. 16-28.

16 Wolff, J.G. (1994), 'Towards a new concept of software', *Software Engineering Journal* 9 (1), pp. 27-38.

17 Wolff, J.G. (1994), 'Computing and information compression: a reply', *AI Communications* 7 (3/4), pp. 203-219.

18 Wolff, J.G. (1993), 'Computing, cognition and information compression', *AI Communications* 6 (2), pp. 107-127.

19 Wolff, J.G. (1991), *Towards a Theory of Cognition and Computing*. Chichester, England: Ellis Horwood.

20 Wolff, J.G. (1990), 'Simplicity and power - some unifying ideas in computing', *Computer Journal* 33 (6), pp. 518-534.

21 Wolff, J.G. (1988), 'Learning syntax and meanings through optimization and distributional analysis', in Y. Levy, I. M. Schlesinger and M. D. S. Braine, eds., *Categories and Processes in Language Acquisition.* Hillsdale, New Jersey: Lawrence Erlbaum, pp. 179-215, Chapter 7. (Reproduced in Ref. 19, Chapter 2).

22 Wolff, J.G., (1982), 'Language acquisition, data compression and generalisation', *Language & Communication. 2*, pp. 57-89. (Reproduced in Ref. 19, Chapter 3)