# PROBABILISTIC REASONING AS INFORMATION COMPRESSION BY MULTIPLE ALIGNMENT, UNIFICATION AND SEARCH (I): INTRODUCTION

J Gerard Wolff

University of Wales, Bangor, UK
gerry@sees.bangor.ac.uk

**Abstract:** This article and its two accompanying articles introduce the idea that probabilistic reasoning (PrbRs) may be understood as information compression by multiple alignment, unification and search (ICMAUS).

In this context, *multiple alignment* has a meaning which is similar to but distinct from its meaning in bio-informatics, while *unification* means a simple merging of matching patterns, a meaning which is related to but simpler than the meaning of that term in logic.

This article, the first of the three, describes the concept of multiple alignment as it has been developed in this research, illustrated with a simple example. Also described is the way in which information compression associated with a multiple alignment may be calculated.

A software model, SP61, has been developed for the discovery and formation of 'good' multiple alignments, evaluated in terms of information compression. The organisation and operation of this model is described in outline.

As an introduction to the two accompanying articles, this article describes in outline how multiple alignments may provide a basis for PrbRs.

**Key Words:** Probabilistic reasoning; multiple alignment; unification; information compression.

**Category:** Computing Methodologies/Artificial Intelligence/Reasoning (uncertainty, 'fuzzy' and probabilistic). I.2.SD.1.2.3.

## 1 Introduction

Quoting Benjamin Franklin ("Nothing is certain but death and taxes"), Ginsberg [16] writes (p. 2) that: "The view that Franklin was expressing is that virtually every conclusion we draw [in reasoning] is an uncertain one." He goes on to say: "This sort of reasoning in the face of uncertainty... has ... proved to be remarkably difficult to formalise."

This article is the first of three articles introducing the idea that *probabilistic reasoning* (PrbRs) may possibly be understood as a process of *information compression* (IC) by *multiple alignment* with *unification* and *search* (ICMAUS).

In this context, PrbRs will be used quite loosely to mean any kind of reasoning which attempts to allow for uncertainty of any kind. This includes kinds of reasoning which use default values for variables where specific values are unknown and kinds of 'nonmonotonic' reasoning which allow probabilistic conclusions reached on the strength of one state of knowledge to be overturned when new knowledge is acquired.

In this context, *multiple alignment* (MA) has a meaning which is similar to but distinct from its meaning in bio-informatics while *unification* means a simple merging of matching patterns, a meaning which is related to but simpler than the meaning of that term in logic.

The term *search* in this context means the systematic exploration of the abstract space of possible alignments, normally constrained in some way (using heuristic techniques or otherwise) to achieve useful results in realistic timescales.

### 1.1 Content

This article introduces these proposals and the research programme in which they have been developed. The concept of MA is described as it has been developed in this research. The way in which the IC associated with any alignment may be calculated is described, first in qualitative terms with reference to an example and then in more precise terms. This is followed by a description of SP61, a software model designed to discover and construct MAs which are 'good' in terms of IC. Finally, this article describes in outline how multiple alignments may provide a basis for PrbRs.

The second article begins by showing how the probabilities of inferences derived from any alignment may be calculated. The remainder of this article provides a range of examples showing how probabilistic inferences may be formed using ICMAUS. Examples include:

1 PrbRs associated with probabilistic and fuzzy pattern recognition and information retrieval. Associated topics include inheritance of attributes in a class hierarchy or heterarchy and the recognition of 'polythetic' categories.
2 One-step 'deductive' reasoning.
3 The integration of 'deductive' and 'abductive' reasoning within a single framework.
4 Chains of reasoning:
   - Reasoning with networks or trees (but excluding Bayesian networks - see below).
   - Reasoning with 'rules'.

The third article presents further examples showing how other kinds of PrbRs may be understood as ICMAUS. These include:

1 Hypothetical ('what if') reasoning.
2 'Indirection' in information retrieval.
3 Reasoning with default values.
4 Nonmonotonic reasoning.
5 Solving geometric analogy problems.
6 Causal reasoning and the phenomenon of 'explaining away': ICMAUS as a possible alternative to Bayesian networks.

The article ends with a brief review of how these proposals may be evaluated and some concluding remarks.

Definitions of terms used in this article and the other two are presented in Appendix A1.

### 1.2 'Reasoning' and 'inference'

Before we proceed, some clarification is necessary for the meanings of the terms *reasoning* and *inference* as they will be used in this article.

The term 'reasoning' will be used to mean any process of "going beyond the information given". In most forms of 'deductive' reasoning, conclusions are either 'true' or 'false', whereas in 'probabilistic reasoning' (PrbRs), conclusions are rarely totally true or false but have some kind of 'probability' attaching to them, e.g. "Black clouds mean that it will probably rain".

All-or-nothing 'deductive' reasoning as it appears in logic will not be considered in this article, except in passing. However, given Gödel's demonstration that there is irreducible uncertainty associated with all but the simplest kinds of formal system, it seems that deductive reasoning might usefully be treated as probabilistic reasoning where probability values are close to 0 or 1. Exploring the implications of that idea would take us well beyond the scope of this article.

As it is generally used, the term 'inference' has three distinct but related meanings:

- It is sometimes used to mean a process of constructing a grammar or other kind of knowledge structure by 'induction' from a body of 'raw' data.
- It is also often used with essentially the same meaning as probabilistic reasoning as described above.
- It may also be used to refer to the *result* or *product* of a process of PrbRs.

Only the second and third of these three meanings will be used in this article although it is anticipated that, in research still to be done, the ICMAUS concepts may be extended with relatively little modification to model unsupervised grammatical inference and similar processes of unsupervised inductive learning.[1]

### 1.3 Background and context

The proposals in these articles have been developed within a programme of research developing the 'SP' conjecture that:

> *All kinds of computing and formal reasoning may usefully be understood as information compression by multiple alignment, unification and search,*

and developing a 'new generation' computing system based on this thinking.[2]

A subsidiary conjecture, which is implied by the main conjecture, is that

> *All kinds of information compression may be understood as pattern*

---

1. As noted in Section 1.3, the application of ICMAUS concepts to learning has not yet been seriously attempted but this entire research programme originated in earlier research on language learning (and other kinds of learning) based on similar principles (see [54, 55]).

2. IC may be interpreted as a process of removing unnecessary (redundant) complexity in information - and thus maximising *simplicity* - whilst preserving as much as possible of its non-redundant descriptive *power*. Hence the name 'SP' applied to the central conjecture and other aspects of this research.

*matching, unification and search.*

Thus, as a working hypothesis, it is assumed that well-known techniques for compressing information such as arithmetic coding, wavelet coding, the use of vectors to describe diagrams, and other techniques, may, ultimately, be understood in terms of pattern matching, unification and search. Whether or not this is true is a question which is likely to remain open for some time.

Background thinking for this research programme is described in [52] and [47]. To date, the concepts have been developed in relation to the following fields:

- The whole current research programme is based on an extensive earlier programme of research into the **unsupervised inductive learning of natural languages and non-linguistic cognitive structures** (see [53, 54, 55] and earlier work cited in those sources). That said, the current SP model has not yet been developed to accommodate learning. This is planned as the next main phase in the development of the model
- **Probabilistic Inference**. These three articles are the main source to date but relevant ideas have been developed in [43], (a discussion of causal inferences in this framework), [44] ('logic' as ICMAUS) and [46] (ICMAUS as an integrating framework for different kinds of inference).
- **Best-Match Information Retrieval and Pattern Recognition**. Article [50] describes SP21 which is essentially a version of dynamic programming which can find good partial matches between two patterns and which has advantages compared with standard forms of dynamic programming.
- **Multi-Level Parsing of Natural and Artificial Languages**. A recent version of the model (SP52), described in [42] has been developed as a model of 'parsing' in the sense understood in theoretical linguistics and natural language processing where a one-dimensional 'sentence' or other input pattern is analysed into labelled parts, subparts, subsubparts etc at arbitrarily many 'levels' in an hierarchical grammar.
- **Automation of software design** and the **execution of software functions** [51].
- It has been argued [45] that **the ICMAUS model may be seen as a 'universal' model of computing** which may be regarded as an interpretation and augmentation of the Turing 'universal' model of computing and equivalent models such as Post's 'Canonical System'.
- Although it has not yet been the subject of any article or report in this research programme, it should be said that the whole framework of ideas embraces **a theory of knowledge representation**. In brief, this theory is that:
    - All kinds of knowledge may be represented by *patterns* (of one or more dimensions) as defined in Appendix A1.
    - Directly or indirectly, all kinds of knowledge about the 'world' may be derived by ICMAUS from 'raw' data about the 'world'.

Regarding the last point, the way in which the structure of natural languages may be represented as patterns is described in [42]. Many of the ideas in that article may be applied to the representation of non-linguistic knowledge. Article [40] shows how some other kinds of knowledge structure (networks, trees and rules; and class hierarchies and heterarchies with inheritance of attributes) may be accommodated in the ICMAUS framework.

Again, with regard to the idea that all emperical knowledge is derived from 'raw' data by inductive learning, an implication is that, in using any kind of knowledge which purports to be empirical, we should take account of the fact that this knowledge is derived from raw data which had spatial or temporal dimensions, or both. Some or all of this information about spatial and/or temporal dimensions may have been lost (e.g., when 2D information is approximated using 1D patterns) but we should avoid using the abstracted information in any way that is inconsistent with the 2-, 3-, or 4-dimensional origins of the information.

### 1.3.1 Potential benefits of integration

If, as conjectured, it proves possible to understand a wide variety of concepts and phenomena in cognition and computing in terms of ICMAUS, this should bring the kinds of benefits which one would expect from any (good) scientific theory:

- Simplification of one's thinking about the phenomena covered by the theory. Instead of using a multiplicity of concepts which have little or no connection with each other, one need deal only with the concepts in the theory and these should be relatively simple.
- New insights and predictions of phenomena beyond the scope of the concepts and observations from which the theory was derived. Some of these possibilities are discussed in Chapter 6 of [53].

In terms of practicalities, the anticipated benefits of an SP system include:

- The expectation that the relatively sophisticated search mechanisms in the SP system will be applicable across a wide range of applications thus saving the need for the re-programming of equivalent mechanisms in each of those applications. This means:
  - Lower costs in developing and maintaining diverse systems.
  - Greater 'quality', especially reliability, because the opportunities to introduce bugs will be reduced and there is a better chance of weeding out bugs in a single core system instead several diverse systems.
- Provision of a generic core for diverse applications should facilitate the integration of those applications and should reduce incompatibilities between applications.

## 1.4 Related research and novelty of the present proposals

As an attempt to integrate concepts across several areas of computing, the SP programme naturally has many connections with other research. Some of these connections are described in [53, 52, 48].

The closest links are with work on Algorithmic Information Theory (AIT, see, for example, [22]) and Minimum Length Encoding (MLE). The latter is an umbrella term for Minimum Description Length encoding (MDL) and Minimum Message Length encoding (MML) - see, for example, [31, 37, 28, 4, 13]. MLE is itself closely related to Bayesian inference (see, for example, [7, 26]).

### 1.4.1 The key idea in AIT and MLE

The MLE principle may be regarded as a relatively precise statement of Occam's Razor. It appears that the MLE principle was first proposed by Solomonoff [31] in relation to 'inductive inference', meaning the unsupervised 'learning' of a grammar or similar knowledge structure from 'raw' data (see Section 1.2). In that context, the key idea is that learning ('inference') may be seen as lossless compression of 'raw' data and that, ***in measuring compression, we must measure the size of the raw data in its encoded form together with the size of the code patterns which are needed to create the encoding***.

Using the total of code size (C) and size of encoded data ($D_C$) appears to give us a handle on some subtle problems in inductive inference and in the related problem of understanding how children learn a first language. With regard to the latter, we need to understand how children can learn to discriminate 'correct' generalisations of grammatical rules from incorrect generalisations (e.g., "I was hitted") when, by definition, both kinds of generalisation have zero frequency in the child's experience. Given that there is evidence that children do not depend on adults for the correction of overgeneralisations, MLE provides a neat explanation of how the two kinds of generalisation can be discriminated: 'correct' generalisations yield an overall reduction in ($C + D_C$) via a reduction in the size of C without a corresponding increase in $D_C$, while 'incorrect' generalisations yield an overall increase in ($C + D_C$) via a reduction in C which is more than outweighed by an increase in $D_C$ (see [54]).

### 1.4.2 Research on probabilistic reasoning

There is now a huge literature on PrbRs. A well-known authoritative survey of the field, with an emphasis on Bayesian networks, is provided by [24] although this source is now, perhaps, in need of updating. A useful review from the same year is [17].

A more recent collection of articles, which together provide a broad coverage of the subject, appears in [13]. A relatively short but useful review of "uncertainty handling formalisms" is provided by [24]. Regarding the application of different kinds of 'logic' to nonmonotonic and uncertain reasoning, there is a mine of useful information in the articles in [13] covering such things as 'default logic', 'autoepistemic logic', 'circumscription', 'defeasible logic', 'uncertainty logics' and 'possibilistic logic'. In that volume, the chapter by Ginsberg [16] provides an excellent introduction to the problems of nonmonotonic reasoning.

Other recent articles in this area include [31], [20], [32], [20], [8] and [5].

### 1.4.3 Information compression and probabilistic reasoning

Naturally enough, much of the literature on probabilistic reasoning deals directly with concepts of probability, especially conditional probability. Since, however, there is a close connection between probability and compression (mediated by coding schemes such as the Huffman coding scheme - see, for example, [8] - or the Shannon-Fano-Elias coding scheme, *ibid.*), concepts of probability imply corresponding concepts of compression.

That said, a primary emphasis on compression rather than probability provides an alternative perspective on the subject which may prove useful. Apart from the present work, relevant sources in that connection include [31], [37], [38], [36], [22], [9], [17] and [18].

### 1.4.4 Distinctive features of this research

In relation to other research in PrbRs, the most distinctive features of the present proposals appear to be:

- Part of the foundation of this research is the conjecture that all kinds of regularity (redundancy) in the world may be expressed as *patterns*[3] of entities or events (in one or more dimensions) which co-occur with a relatively high frequency [47]. Coupled with the idea that all kinds of redundancy may be reduced or eliminated by the unification of matching patterns of symbols (including subsequences of larger patterns which may be coherent or discontinuous within those larger patterns), this conjecture seems to be distinctive of the research programme.
- The derivation of all probability values from absolute frequencies of patterns contrasts with the storage of conditional probabilities which is a feature of Bayesian networks and other systems for PrbRs (see, for example, [25]).
- The idea that ICMAUS may provide a framework for understanding PrbRs and integrating different kinds of PrbRs appears to be entirely novel. It seems that, currently, no other researchers are developing this idea.
- The idea that, in addition to integrating different kinds of PrbRs, the ICMAUS framework may integrate concepts of PrbRs with several other kinds of "computing and formal reasoning" seems to be distinctive of this research.
- The concept of MA as it has been developed in this research differs, in a non-trivial way, from MA as it is normally conceived in bio-informatics (see Section 2.1).

## 1.5 Scope and orientation of these articles and of the research programme

The primary purpose of these three articles is to present the ICMAUS proposals and attempt to demonstrate their usefulness for understanding PrbRs. Space does not permit any detailed comparison between the ICMAUS proposals for PrbRs and alternative systems such as Markov or Bayesian networks, the Dempster-Shafer theory, and others (see, for example, [13]; [24]).

It cannot be emphasised too strongly that ***the overall goal of this research programme is the integration of diverse ideas across subfields of cognition and computing***. As with any good theory in science, the unifying framework which this research aims to develop should be as ***simple*** as possible, without sacrificing descriptive or explanatory ***power***. Thus *simplicity* and *power* are the mainstay of both the SP theory and its evaluation.

---

3. For definitions of terms, see Appendix A1.

## 1.6 Theory and applications

Although the abstract model which is being developed is expressed in a software model (SP61, described in Section 6) the primary purpose in developing this working system has been to provide a 'test bed' for the abstract model and a vehicle for demonstrating the concepts as they have been developed to date. SP61 is *not* intended as a system to be used in practical applications and should not be evaluated as such.

Although the primary focus of the current work is the development of an abstract model, the need to avoid insuperable obstacles to practical applications in the future has at all times been born in mind.

In particular, the development of the model has, at all times, been conditioned by the need to ensure that the computational processes in the model would be scaleable to large applications. As noted in Section 6.1, the estimated 'big O' values for the time complexity and space complexity of the model are satisfactory, especially if high levels of parallel processing are applied.

As indicated above, it is envisaged that mature versions of the SP system will exploit high levels of parallel processing to ensure that there is sufficient power to drive the relatively 'expensive' search for 'good' matches which is at the heart of the model. At all times, this objective has been kept in view to ensure that the model can be implemented within a high-parallel environment when that is required.

## 2 Multiple alignment problems

The term *multiple alignment* is normally associated with the computational analysis of (symbolic representations of) sequences of DNA bases or sequences of amino acid residues as part of the process of elucidating the structure, functions or evolution of the corresponding molecules. The aim of the computation is to find one or more alignments of matching symbols in two or more sequences which are, in some sense, 'good'. Possible meanings for that term are discussed in Section 4, below. An example of an alignment of DNA sequences is shown in Figure 1.

-------------------------------------------------------------------

Figure 1 about here

-------------------------------------------------------------------

**Figure 1**

A 'good' alignment amongst five DNA sequences (adapted from Fig. 6 in [29], with permission from Oxford University Press).

In this area of research, it is widely recognised that the number of possible alignments of symbols is normally too large to be searched exhaustively and that, to achieve a search which has acceptable speed and acceptable scaling properties, 'heuristic' techniques must normally be used. Heuristic techniques include 'hill climbing' (sometimes called 'descent'), 'beam search', 'genetic algorithms', 'simulated annealing', 'dynamic programming' and others. With these techniques, searching is done in stages, with a progressive narrowing of the search in successive stages using some kind of measure of goodness of alignments to guide the search. These techniques may be described generically as ‘metrics-guided search’.

With these techniques, ideal solutions cannot normally be guaranteed but acceptably good approximate solutions can normally be found without excessive computational demands.

There is now a fairly large literature about methods for finding good alignments amongst two or more sequences of symbols. Some of the existing methods are reviewed in [35, 3, 5, 11]. Because of the way in which the concept of MA has been generalised in this research (see next), none of the current methods for finding MAs are suitable for incorporation in the proposed SP system. Hence the development of a new method, outlined in Section 6

### 2.1 Generalisation of the concept of multiple alignment

In this research, the concept of MA has been generalised in the following way:

1 One (or more) of the sequences of symbols to be aligned has a special status and is designated as ‘New’. The way in which the concept of ‘New’ appears to relate to established concepts in computing is shown in Table 1.

2 All other sequences are designated as ‘Old’. The way in which the concept of ‘Old’ appears to relate to established concepts in computing is also shown in Table 1.

3 A ‘good’ alignment is one which, through the unification of symbols in New with symbols in Old, and through unifications amongst the symbols in Old, leads to a relatively large amount of compression of New in terms of the sequences in Old. How this may be done is explained in Section 4, below.

4 An implication of this way of framing the alignment problem is that, by contrast with ‘multiple alignment’ as normally understood in bio-informatics, any given sequence in Old may appear two or more times in any one alignment and may therefore be aligned with itself (with the obvious restriction that any one instance of a symbol may not be aligned with itself).[4]

---

4. With the kind of MA shown in Figure 1, it is obviously possible to include two or more *copies* of a given sequence in any one alignment. To my knowledge, this is never done in practice because it would simply lead to the trivial alignment of each symbol in one copy with the corresponding symbol or symbols in the one or more other copies. What is proposed for the ICMAUS framework is different: any *one* pattern may appear two or more times in an alignment. Each appearance is just that, it is an *appearance* of *one* pattern, not a duplicate *copy* of a pattern. Since each appearance of a pattern represents the same pattern, it makes no sense to match a symbol from one appearance with the corresponding symbol in another appearance because this is simply matching one instance of symbol with itself. Any such match is spurious and must be forbidden.

| *Area of application* | *New* | *Old* |
|---|---|---|
| Unsupervised inductive learning | 'Raw' data. | Grammar or other knowledge structure created by learning. |
| Parsing | The sentence to be parsed. | The grammar used for parsing. |
| Pattern recognition and scene analysis | A pattern to be recognised. or scene to be analysed. | The stored knowledge used to recognise one pattern or several within a scene. |
| Databases | A 'query' in SQL or other query language. | Records stored in the database. |
| Expert system | A 'query' in the query language for the expert system. | The 'rules' or other knowledge stored in the expert system. |
| Computer program | The 'data' or 'parameters' supplied to the program on each run. | The computer program itself. |

**Table 1**
The way in which the concepts of 'New' and 'Old' in this research appear to relate to established concepts in computing.

### 2.1.1 Higher dimensions

It should be clear that this concept of MA (and the bio-informatics version of the concept) may be generalised to two-dimensional (or even higher-dimensional) patterns. There is likely to be a case, at some stage in the SP programme, for extending the ideas described in this article into the domain of two or more dimensions.

## 3 A simple example

The main concepts to be presented can probably best be described with reference to a simple example. Since 'parsing' in the sense understood in theoretical linguistics and natural language processing has come to be a paradigm for the several concepts to be described, it will provide our first example despite the fact that, when the input sentence or phrase to be parsed is complete, there is no significant PrbRs as understood in this article.

Sections 3.1 and 3.2 show how the parsing of a *very* simple sentence with a *very* simple grammar may be understood as MA. Much more elaborate examples can be found in [42].

### 3.1 Representing a grammar with patterns of symbols

Figure 2 shows a simple context-free phrase-structure grammar (CF-PSG) describing a fragment of the syntax of English. This grammar generates the four sentences '`j o h n r u n s`', '`j o h n w a l k s`', '`s u s a n r u n s`', and '`s u s a n w a l k s`'. Any of these sentences may be parsed in terms of the grammar giving a labelled bracketing like this:

```
(S(N j o h n)(V r u n s))
```

or an equivalent representation in the form of a tree.

```
S → N V
N → j o h n
N → s u s a n
V → w a l k s
V → r u n s
```

**Figure 2**

A CF-PSG describing a fragment of English syntax.

Figure 3 shows the grammar from Figure 2 expressed as a set of *strings*, *sequences* or *patterns* of *symbols*.[5] Each pattern in this 'grammar' is like a re-write rule in the CF-PSG notation except that the rewrite arrow has been removed, some other symbols have been introduced ('`0`', '`1`' and symbols with an initial '`#`' character) and there is a number to the right of each rule.[6,7,8]

The number to the right of each rule in Figure 3 is an imaginary frequency of occurrence of the rule in a parsing of a notional sample of the language. These frequencies of occurrence will be discussed later.

The reasons for the symbols which have been added to each rule will become clear

5. For definitions of terms, see Appendix A1.

6. For the remainder of this article, quote marks will be dropped when referring to any grammar like that in Figure 3 which is expressed as patterns of symbols. Likewise, the word 'rule' with respect to this kind of grammar will be referred to without quote marks.

7. This example of a grammar and of how it is used in parsing may give the impression that the ICMAUS framework is merely a trivial variation of familiar concepts of context-free phrase-structure grammar (CF-PSG) with their well-known inadequacies for representing and analysing the 'context sensitive' structures found in natural languages. The examples presented in [42] show that the ICMAUS framework is much more 'powerful' than CF-PSGs and can accommodate quite subtle context-sensitive features of natural language syntax in a simple and elegant manner. Achieving this expressive power with a relatively simple notation is made possible by the relatively sophisticated search processes which lie at the heart of the SP model.

but a few words of explanation are in order here. The symbols '0' and '1' have been introduced to differentiate the two versions of the 'N' patterns and the two versions of the 'V' patterns. They enter into matching and unification in exactly the same way as other symbols. Although the symbols are the same as are used in other contexts to represent numbers they do not have the meaning of numbers in this grammar.

```
S N #N V #V #S              1000
N 0 j o h n #N              300
N 1 s u s a n #N            700
V 0 w a l k s #V            650
V 1 r u n s #V              350
```

**Figure 3**

A simple grammar written as patterns of symbols. For each rule, there is a number on the right representing the frequency of occurrence of the rule in a notional sample of the language.

The symbols which begin with '#' (e.g., '#S', '#NP') serve as 'termination markers' for patterns in the grammar. Although their informal description as 'termination markers' suggests that these symbols are meta symbols with special meaning, they have no hidden meaning and they enter into matching and unification like every other symbol.[9]

In general, all the symbols which can be seen in Figure 3 enter into matching and unification in the same way. Although some of these symbols can be seen to serve a distinctive role, there is no hidden meaning attached to any of them and there is no formal distinction between upper- and lower-case letters or between digit symbols and alphabetic symbols and so on.[10]

## 3.2 Parsing as alignment of a sentence and rules in a grammar

Figure 4 shows how a parsing of the sentence 'j o h n r u n s' may be seen as an alignment of patterns which includes the sentence and relevant rules from the grammar shown in Figure 3. The similarity between this alignment and the conventional parsing may

---

8. For the sake of clarity in exposition and to save space, all the grammars shown in this article are much simpler than in any practical system. For similar reasons, all examples of MAs which are presented have been chosen so that they are small enough to fit on one page without resorting to font sizes which are too small to read. However, for the reasons given in Section 6.3, the model appears to be general and scalable to realistically large knowledge structures and alignments.

9. See qualification of this assertion below.

10. The foregoing assertions are not strictly true of the method of evaluating alignments which is used in the SP61 model (see Appendix 5.5). The principle of "no meta symbols" and thus "no hidden meanings for symbols" is an ideal which this research aims to attain. But, as a temporary solution to the problem of scoring alignments pending something better, a distinction has been recognised in the method for scoring alignments in the SP61 model (Section 4) between symbols which begin with '#' and all other symbols.

be seen if the symbols in the alignment are 'projected' on to a single sequence, thus:

`S N 0 j o h n #N V 1 r u n s #V #S`

In this projection, the two instances of '`N`' in the second column of the alignment have been merged or 'unified' and likewise for the two instances of '`#N`' in the eighth column and so on wherever there are two or more instances of a symbol in any column.

-------------------------------------------------------------------

Figure 4 about here

-------------------------------------------------------------------

**Figure 4**

Parsing of the sentence '`j o h n r u n s`' as an alignment amongst sequences representing the sentence and relevant rules in the grammar in Figure 3.

This projection is the same as the conventional parsing except that '`0`' and '`1`' symbols are included, right bracket symbols ('`)`') are replaced by 'termination markers' and each of the upper-case symbols is regarded both as a 'label' for a structure and as a left bracket for that structure.

As was noted in Section 2.1, the sentence or other sequence of symbols to be parsed is regarded as New, while the rules in the grammar are regarded as Old. For the sake of readability and ease of interpretation, New is normally placed at the top of each alignment with patterns from Old below it.

For the sake of clarity in Figure 4 and other alignments shown later, each appearance of a pattern in any alignment is given a line to itself. Apart from the convention that New is always at the top, the order in which patterns appear (from top to bottom of the alignment) is entirely arbitrary. An alignment in which the patterns appear in one order is entirely equivalent to an alignment in which they appear in any other order, provided all other aspects of the alignment are the same.

## 4 Evaluation of an alignment in terms of IC

Intuitively, a good alignment is one which has many *hits* (positive matches between symbols), few *gaps* (sequences of one or more symbols which are not part of any hit) and, where there are gaps, they should be as short as possible. It is possible to use measures like these directly in computer programs for finding good MAs and, indeed, they commonly are. However, our confidence in the validity of measures like these may be increased if they can be placed within a broader theoretical framework.

Concepts of information, IC and related concepts of probability may provide a suitable framework. Work on the evaluation of MAs in this tradition includes [27], [12], [2], [5], [1] and [50].

As was indicated in Section 2.1, *a good alignment is defined here as one which provides a basis for an economical coding of New in terms of the patterns Old.*

The compression method which is described in Section 4.2, below, exploits the elementary principle that a (relatively long) sequence of symbols which repeats two or more times in a body of information may be replaced by a shorter 'code', 'codeword' or 'tag pattern' associated with that pattern in some kind of 'dictionary' of patterns. In effect, each instance of the pattern in the data is unified with the same pattern as it appears in the repository of patterns. This is the basis of all standard methods for IC [34].

The present proposals differ from standard methods in two ways:

- Each code serves a dual role: to identify the corresponding pattern uniquely within the grammar, and to mark the left and right ends of the pattern. For present purposes, the second role is required to remove the ambiguity which would otherwise exist about left-to-right sequencing of symbols in alignments.[11]
- In the present scheme, the coding principle may be applied through two or more 'levels' so that the symbols which encode a sequence of two or more patterns at one level may themselves be recognised as an instance of a recurrent pattern which has its own code at the next higher level. Examples will be seen later.

A key point in this connection is that a recurrent pattern may be *discontinuous* in the sense that *the symbols in the pattern are not necessarily contiguous as they appear in any or all of its occurrences*. In other words, a recurrent pattern may appear as a subsequence within larger patterns.[12] Thus, for example, a sequence of symbols like 'A B C D E F' may be recognised as a recurrent pattern within a set of instances which includes patterns like '`P A B Q C R D E F S`', '`A L B C D M N E F O P`', '`X A B C D Y E F Z`' and so on.

In Section 4.2, below, I shall give an informal explanation of the method of calculating the IC associated with any alignment. The details of the method of calculation are presented in Section 5.

## 4.1 Lossless and lossy compression

In connection with any kind of IC, it is important to distinguish 'lossless' compression from 'lossy' compression. The former means IC where redundant information is the only information which is removed. The latter means IC where redundant information may be and normally is removed but, in addition, there is loss of non-redundant information. In the first case, the original information may be re-created with complete fidelity. In the second case, the original information can never be perfectly restored.

Currently, this research programme assumes that lossless compression is the rule. In future research on the ICMAUS concepts, there may be a case for exploring the several ways in which lossy compression may be used.

## 4.2 Evaluation of an alignment in terms of IC: informal description

The following subsections describe informally how the IC associated with an alignment

---

11. Although standard methods of data compression appear, superficially, to use codes without termination markers, there is an unavoidable need for information about the left and right boundaries of each coded pattern and this information is normally provided implicitly by the use of space characters or end-of-line characters or something similar, or by the use of a trie for the recognition of variable-length codes.

12. The terms *subsequence* and *substring* are defined in Appendix A1.

may be calculated when the alignment may be interpreted as a parsing of New in terms of Old. The alignment in Figure 4 will be our example.

The IC associated with any alignment is not realised until New has been encoded in terms of the patterns from Old which appear in the alignment. Nevertheless, it will be convenient at times in this article and the ones that follow, to say that a given alignment encodes New or compresses New. Wherever this form of words is used it should be understood to mean that a code which expresses New in a compressed form may be derived from the alignments under consideration.

### 4.3 Encoding individual symbols

Regarding individual symbols in the sentence and the grammar, the simplest way to encode them is with a 'block' code using a fixed number of bits for each symbol. In the grammar in Figure 3, there are 20 symbol types so the minimum number of bits required for each symbol is $\lceil \log_2 20 \rceil = 5$ bits per symbol.

In fact, the SP61 model uses variable-length codes for symbols, assigned in accordance with a modified version of the Shannon-Fano-Elias (S-F-E) coding scheme[13] so that the shortest codes represent the most frequent symbols and *vice versa*.[14]

Notice that the number of bits required for each symbol is entirely independent of the number of characters in the name of the symbol as it is shown in the examples. Names of symbols are chosen purely for their mnemonic value and to aid comprehension.

There are many variations and refinements that may be made at this level but, in general, the choice of coding system for individual symbols is not critical for the principles to be described below where the focus of interest is the exploitation of redundancy which may be attributed to *sequences* of two or more symbols rather than any redundancy attributed to unbalanced frequencies of individual symbols.

For reasons which are given in Section 5 (and Section 6 of [42]), the code for each symbol has two different values (in bits): a *minimum cost* which is the theoretical minimum number of bits needed to represent that symbol according to the (modified) S-F-E calculations, and an *actual cost* which is larger by a constant *cost factor*, whose value is typically about 10 or 20.

In the following informal description of the encoding principles, the distinction between the minimum cost and the actual cost of each symbol is not important and will be ignored. For the sake of simplicity in this presentation, it will be assumed that all symbols are encoded with the same number of bits so that 'one symbol' can be treated as the minimum unit of information.

---

13. The coding scheme (without the present modification) is described by [8].

14. The Huffman coding scheme, described in [8] and also in [34], is better known than the S-F-E scheme and is a little more efficient in terms of IC. But the S-F-E scheme allows probabilities to be inferred more exactly from the sizes of codes assigned by the scheme as will be described in Section 7.

### 4.4 Encoding words

A word in the sentence to be parsed may be encoded using some shorter code which is associated with the word in the dictionary and which identifies the word uniquely within the dictionary. In this case, the grammar in Figure 3 plays the part of the dictionary of segments in a standard compression system.

In accordance with the remarks above, the code pattern must show where the word starts and also where it finishes. In some cases, this can be achieved by using letter symbols from the word itself. Thus '`g l`' could be used as an abbreviation for '`g i r l`' very much in the same way as we use abbreviations like 'B'ham' for 'Birmingham' in ordinary writing. However, because of the haphazard nature of such markers, there are advantages if we use code symbols introduced especially for the purpose at the beginnings and ends of words.

In the light of these remarks, it should be clear that the word '`j o h n`' in the grammar shown in Figure 3 may be encoded as '`N 0 #N`', the word '`r u n s`' may be encoded as '`V 1 #V`' and likewise for the other words. In all cases there is a modest saving of one or two symbols for each word. If the grammar contained words with fewer than three letters, then the 'saving' in those cases would be negative!

### 4.5 Encoding the sentence

Given the two words in their encoded forms ('`N 0 #N`' for '`j o h n`' and '`V 1 #V`' for '`r u n s`'), the whole sentence may be encoded as '`N 0 #N V 1 #V`'. This code for the sentence achieves a small amount of compression because the code contains 2 symbols less than the number of symbols in the original sentence.

However, this sequence contains the subsequence '`N #N V #V`' and this sequence is a substring within the 'sentence' pattern '`S N #N V #V #S`' - which appears as the first pattern in the grammar. So we may replace the sequence '`N #N V #V`' by the 'code' sequence '`S #S`'. To discriminate the words in this sentence we must add the symbols '`0 1`' from the sequence '`N 0 #N V 1 #V`'. The overall result is an encoded representation of the sentence as:

```
S 0 1 #S.
```

The 4 symbols in this encoding of the sentence represents a useful compression compared with the 8 symbols in the unencoded sentence.

### 4.6 Encoding patterns with arbitrarily many levels of parts and subparts

The simple example we are considering contains only one main part (the whole sentence) and two levels of subparts (the words within the sentence and the symbols within each word). It should be clear that the principles which have been described can be applied through arbitrarily many levels in a part-whole hierarchy. Examples are presented in [42].

### 4.7 Discussion

Each pattern expresses sequential redundancy in the data to be encoded and this sequential redundancy can be exploited to reduce the number of symbols which need to be written out explicitly. In the grammar shown in Figure 3, each pattern for an individual word expresses the sequential redundancy of the letters within that word. The pattern for a sentence expresses the sequential redundancy of the pattern: 'noun' followed by 'verb'.

Since this principle operates at all levels in the 'hierarchy' of patterns, many of the symbols at intermediate levels may be omitted completely. A sentence may be specified with symbols marking the start and end of the sentence pattern together with interpolated symbols which discriminate amongst alternatives at lower levels.

The method which has been described illustrates the role of context in the encoding of information. Any one symbol like '0' or '1' is ambiguous in terms of the patterns in the grammar in Figure 3. But in the context of the pattern 'S 0 1 S', it is possible to assign each instance of '0' or '1' unambiguously to one of the words in the grammar, giving the sequence of words in the original sentence.

### 4.8 'Gaps' and 'learning'

Consider the simple alignments shown in Figure 5. In (a) there is a gap in the sequence of hits in the pattern from Old while in (b) the gap in the sequence of hits is in the New pattern. In alignment (c), there is a gap in both New and Old.

These three alignments differ from the one shown in Figure 4 because, in every case, assuming lossless compression, the pattern from New cannot be encoded precisely in terms of the pattern from Old which appears in the alignment. In broad terms, there seem to be two kinds of solution to this problem.

-------------------------------------------------------------------

Figure 5 about here

-------------------------------------------------------------------

**Figure 5**

Some examples of 'gaps' in alignments.

#### 4.8.1 Taking account of gaps by 'learning'

In future versions of the SP model, it is anticipated that the existence of 'gaps' like the ones just shown will be accommodated by creating new patterns and codes and by modifying the patterns in Old. This may be regarded as a form of unsupervised 'learning'.

For example, in Figure 5 (a), the pattern from Old may be modified to become something like '`t h e A #A t a b l e`' and a new pattern, something like '`A 0 b i g #A`', will be added to Old as a separate entity. Similar changes may be made in the case of alignment (b). For the alignment in Figure 5 (c), the pattern from Old would be changed to something like '`t h e A #A t a b l e`', as before, and two new patterns - something like '`A 0 b i g #A`' and '`A 1 w o o d e n #A`' - would be added to Old.

#### 4.8.2 An alternative 'stop-gap' method for taking account of gaps

Pending the introduction of 'learning' to the SP model, the SP61 model uses an alternative method for taking account of gaps when compression differences are computed. The method exploits a dice rolling analogy together with some 'standard' probability theory presented in [23]. There is insufficient space in this article to describe the method in detail. A brief outline of the method and how it is applied is presented in Section 5, below.

#### 4.8.3 Gaps and compression scores

In general, our intuitions tell us that an alignment with few gaps and/or small ones is 'better' than an alignment with many gaps and/or large ones.

It would take us too far afield to discuss these issues in detail but it seems that both of the methods for taking account of gaps which have been outlined will tend to reduce the sizes of compression differences, broadly in line with our intuitions:

- In the case of 'learning', every gap requires the creation of new codes and these are added to the 'cost' of encoding. So it is reasonably clear that, other things being equal, an alignment with many gaps will have a lower compression difference than one with few gaps. It is less clear why long gaps should reduce compression differences more than small gaps. A possible reason is that, with long gaps, it is less easy to find the 'correct' alignment and there are many 'wrong' alignments which fragment one or more of the long gaps into a larger number of smaller ones.
- In the case of the method used in SP61, the dice-rolling analogy on which it is based has a clear implication that many gaps and long ones will reduce the compression difference.

#### 4.8.4 'Invalid' alignments

All of the foregoing remarks about gaps assumes that the gaps occur in matches between New patterns and Old patterns. However, there can also be gaps in matches between two patterns both of which are from Old. Amongst the latter kind of alignment, there are alignments containing 'mis-matches' between two patterns from Old, as in the two alignments shown here:

```
a   b           a b
|   |           | |
a x b  or this  a b x
|   |           | |
a y b           a b y
```

In each of these two examples, there is ambiguity about the relative left-right positions of 'x' and 'y' which means that it is not possible to unify the alignment into a single sequence without making arbitrary assumptions about whether 'x' precedes 'y' or *vice versa.*

In some future 'learning' version of the SP model, it may be possible to handle this kind of mis-match in the same kind of way as was discussed in Section 4.3.1. However, SP61 assumes that the patterns in Old are immutable. For this reason, it treats all alignments like the two above as 'invalid'. If that kind of mis-match is detected while an alignment is being formed, the alignment discarded and thus excluded from any further processing.

### 4.9 Decoding by compression

Although it is not particularly relevant to PrbRs, we may note in passing that, paradoxical as it may seem, the process of 'decoding' an encoded version of a sentence so that it is expanded to recreate the original sentence may be achieved by *exactly* the same processes of ICMAUS as is used to achieve the original compression. How this can be done is explained in Section 6 of [42].

## 5 Calculating the compression associated with an alignment

The method of calculating the *compression difference* (CD) associated with an alignment of patterns which was described informally in the last section is summarised in more formal terms here. This is the method embodied in the SP61 model, outlined in Section 6.

The method is designed to calculate the compression of New information (the sentence to be parsed) which may be achieved by encoding New information in terms of Old information (where Old information is the patterns of symbols representing the grammar used in parsing). This CD is calculated as:

$$CD = B_N - B_E$$

where $B_N$ is the number of bits required to represent in 'raw' form (without any encoding except modified S-F-E coding at the level of single symbols) those symbols from New which form hits with symbols from Old in the alignment. $B_E$ is the number of bits required for the encoding of that New information in terms of Old information.[15] How these values are calculated is described below.

### 5.1 Information costs of symbols

If a simple block code is used for symbols, then the *minimum cost*, M, for each symbol is

$$M = \log_2|A|$$

bits where |A| is the number of symbol types in the alphabet of symbol types (A) used throughout New and Old. If we know that a binary alphabet is to be used, M may be rounded up to the nearest integer, thus: $M = \lceil \log_2|A| \rceil$ bits.

#### 5.1.1 Variable-length codes

If there are variations in the frequencies of symbols, as there normally are, we can save space by using variable-length codes for each symbol such as one might construct using Huffman coding (see [8]).

---

15. An obvious alternative to CD as a measure of compression is 'compression ratio' (CR) calculated as $CR = B_E / B_N$. For some purposes, CR may be preferable to CD because it expresses the 'efficiency' of compression in a manner which is largely independent of the absolute sizes of $B_E$ or $B_N$. But, for the purpose of guiding a search for MAs which are 'good' in terms of the compression of New in terms of Old, CR has the disadvantage that it can give spurious prominence to alignments which encode a small part of New efficiently but which neglect the major part of New. The CD measure does not have this bias.

Although Huffman codes are well-known and efficient, they have the disadvantage for present purposes that they can lead to inaccuracies when codes are translated into probabilities (considered in [39]). Codes which are constructed by the Shannon-Fano-Elias (S-F-E) method (see [8]) are slightly less efficient but avoid the inaccuracies which can arise if Huffman codes are translated into probabilities. However, for reasons connected with the practicalities of coding and decoding in a binary environment, the lengths of the codes produced by the S-F-E method are, themselves, slightly too long. For that reason, the SP61 model (Section 6) calculates the size of the code for each symbol type using this formula:

$$\mathrm{M}_i = \log_2(1 / p_i)$$

where $\mathrm{M}_i$ is the minimum cost of the $i$th symbol type and $p_i$ is the probability of the $i$th symbol type. This formula gives a value which is slightly smaller than the size of the code which would be produced by the S-F-E method ($\mathrm{M}_i = \lceil \log_2(1 / p_i) \rceil + 1$), partly because there is no rounding up as there is in the S-F-E method. Rounding up is not used because it assumes the use of a binary alphabet, an assumption which need not be true.

Although the formula is not totally accurate as a measure of minimum code sizes that one could use in practice in a binary environment, it has the advantage for present purposes that it allows probabilities of patterns, symbols and alignments to be derived in a straightforward manner from the minimum cost of the symbols used to encode a pattern or an alignment (as described in [39]).

### 5.1.2 Probabilities and frequencies

For the calculation of $\mathrm{M}_i$, above, $p_i$ is calculated as:

$$p_i = f_i / \mathrm{F}$$

where F is the total frequency of the $|\mathrm{A}|$ symbol types and $f_i$ is the frequency of the $i$th symbol type.

The value of $f_i$ for each symbol type can be determined easily by counting if the 'raw' data is available from which a set of patterns was derived. If only a sample of the raw data is available, frequencies can be estimated, and should be acceptable for most purposes providing that the sample is representative and reasonably large.

If only the derived patterns are available, values for $f_i$ can be calculated as:

$$f_i = \sum_{j=1}^{\mathrm{P}} (f_j \times o_j)$$

where $f_j$ is the (notional) frequency of the $j$th pattern in the grammar (illustrated by the numbers on the right of Figure 3), $o_j$ is the number of occurrences of the given symbol in the $j$th pattern and P is the number of patterns in the grammar.

The formula just given looks innocent enough but it can yield frequency values for symbols which are too high. Consider, for example, a small 'world' containing the sequence of symbols '`A B C`'. It is possible to represent this 'world' with two patterns: '`A B`' and '`B C`', each with a frequency of 1. If the formula, above, is applied to this database, the frequency of '`B`' is calculated as 2, not 1 as it should be.

Although it is implausible in this example to use two patterns instead of one to represent the 'raw' data, comparable situations are quite common in more realistic cases. For example, the frequency of the grammatical symbol '`N`' (for noun) in the parsing example presented in Section 2.1.1 is calculated as 2000 from the frequencies of patterns shown in Figure 3 using the formula above. However, this overlooks the fact that the frequencies shown in the figure are frequencies that would be obtained from (good) parsings of 'raw' data (see Section 3). In these parsings, the '`N`' symbol at the start of each word is unified with the corresponding '`N`' symbol in the sentence pattern but it is counted twice, once in the word pattern and once in the sentence pattern. Thus the correct value for the frequency of '`N`' in this example should be 1000 not 2000 as calculated by the formula.

Although the formula can give frequency values for symbols which are too high in some cases, it has been assumed for the time being that the formula is sufficiently accurate for most practical purposes.

#### 5.1.3 'Minimum' and 'actual' cost for each symbol

Given a value of $M_i$, as described above, the minimum cost of each symbol is multiplied by a *cost factor*, *c*, to obtain an *actual cost*, $A_i$, thus:

$$A_i = M_i \times c \qquad c > 1.$$

The cost factor must be bigger than 1 and is normally about 10 or 20. The purpose of actual costs and appropriate values for the cost factor are discussed in Section 5.3, below.

### 5.2 Calculation of E, the minimum number of bits required for the encoding of a given pattern in Old

The calculation of $B_E$ for any alignment requires a value for the *encoding cost*, E, for each pattern from Old which appears in the alignment.

Since there is a frequency of occurrence associated with each pattern in any grammar, it is possible to calculate a minimum for the value of E for each pattern using the S-F-E method or the formula used to calculate the minimum cost of each symbol type (Section 5.1). However, there is an alternative method of calculating E which, for present purposes, appears to be more useful and which has been adopted in the SP61 model described in Section 6.

In summary, the alternative method is to calculate E as

$$E = \sum_{i=1}^{n} D_i$$

where $D_i$ is the M value for the *i*th symbol in a subsequence of *n* 'discrimination' symbols within the given pattern which identifies the pattern uniquely amongst the patterns in the grammar without over-specifying the pattern.

Ideally, the discrimination symbols for a pattern would be whatever subsequence of the pattern was most distinctive of the pattern, regardless of the position of the symbols within the pattern. However, in the SP61 model, two constraints have been imposed:

- The simplifying assumption has been made that the discrimination symbols are the smallest substring of one or more symbols starting at the beginning of the pattern which enables the pattern to be identified uniquely within the grammar. For any pattern, it is easy to discover what this substring is by a process of systematic comparison of candidate substrings with corresponding symbols in other patterns in the grammar.

Although a constrained subsequence of symbols is used in calculating the value of E for the pattern, this does not mean that a pattern can only ever be recognised by those symbols and no others. In the SP61 model, a pattern can be fully or partially recognised by any subsequence of its symbols.

- For each pattern which ends in a symbol containing the hash character ('#'), this 'termination' symbol is added to the set of discrimination symbols for the pattern if it is not otherwise there.

The second constraint is not entirely satisfactory, not least because it conflicts with the principle that all symbols should have the same status and none should have any 'hidden' meaning. The constraint has been adopted to compensate for the fact that 'termination' symbols are largely redundant for the purpose of identifying patterns because each one duplicates the information in the first symbol of the same pattern (the primary function of 'termination' symbols is to mark the ends of patterns). This redundancy means that, without the constraint, the method of calculating $B_E$ (see Section 5.5, below) can sometimes fail to discriminate amongst alignments which clearly differ in their potential for economical encoding.

## 5.3 Calculation of $B_N$ (the number of bits required to express 'hit' symbols from New in 'raw' form, without encoding)

For any one alignment, $B_N$ is calculated as:

$$B_n = \sum_{i=1}^{h} A_i$$

where $A_i$ is the actual cost of the symbol in New corresponding to the $i$th hit in a sequence of hits, $H_1 \ldots H_h$, with an adjustment to be described in the next paragraph. The hit sequence $H_1 \ldots H_h$ comprises the hits between symbols in New and symbols in patterns in Old. The symbols from New in this hit sequence are a subsequence of the sequence $N_1 \ldots N_n$, which is the pattern in New.

### 5.3.1 The 'cost factor' and the 'actual cost' of symbols

The actual cost of a symbol (described in Section 5.1) is used *only* when it appears in New and its sole purpose is for the calculation of $B_N$, as described above. The reason for introducing this idea and the cost factor described in the same section is that we must suppose that each symbol in New is an analogue of a relatively large 'chunk' of information (e.g., a section of the sound wave of speech) while the corresponding symbol when it appears in Old is an analogue of a relatively short 'code' which represents the corresponding chunk of information in New.

In general, codes must be smaller than the things they encode. If this does not obtain then, clearly, no compression can be achieved.

### 5.4 Allowing for Gaps

Before the above formula is applied, the value of each $A_i$ is adjusted to take account of any 'gap' which may exist between the given hit and any previous hits in the sequence of hits between New and patterns in Old. For this purpose, the alignment is treated as if it were two sequences of symbols: the sequence of symbols which is New (the sentence being parsed) and the sequence of symbols which is the projection of the alignment into a single sequence.

As indicated above, there is insufficient space to present fully the method of allowing for gaps. In outline, it is based on an analogy with the rolling of two A-sided dice, where A is the size of the alphabet used in New and Old. The sequence of rolls of one die corresponds with the sequence of symbols in New and the sequence of rolls of the other die corresponds with the sequence of symbols in the projection. The method is based closely on the method described in [23] (Chapter 3) for calculating probabilities of various contingencies in problems of this type.

For the symbol corresponding to the $i$th hit in the sequence $H_1 \ldots H_h$, the adjusted value of $A_i$ is calculated as:

$$A_i = a_i \times G_s$$

where, $a_i$ is the actual cost of the symbol corresponding to the $i$th hit in $H_1 \ldots. H_h$, and $G_s$ is the $s$th entry in a table of 'scaling factors' which is calculated at the outset of processing. The value of $G_1$ is always 1.

For each hit in $H_1 \ldots. H_h$ after the first, the variable $s$ (which represents the 'span' between the current hit in $H_1 \ldots H_h$ and the preceding hit) is calculated as:

$$s = (P_i - P_{i-1}) \times (C_i - C_{i-1})$$

where $P_i$ is the position in $N_1 \ldots N_n$ of the symbol corresponding to the $i$th hit in $H_1 \ldots H_h$, $P_{i-1}$ is the position in $N_1 \ldots N_n$ of the symbol corresponding to the $(i$-1)th hit in $H_1 \ldots H_h$. $C_i$ and $C_{i-1}$ are the analogous positions in the projection of the alignment into a single sequence - which means that $C_i$ and $C_{i-1}$ represent columns in the alignment itself.

### 5.5 Calculation of $B_E$ (the number of bits required for the encoding of New information in terms of Old information)

In the long run, the most satisfactory way of calculating $B_E$ (the number of bits required to encode an alignment) is simply to create an explicit code from the alignment in the manner described in Sections 4.2.2, 4.2.3 and 4.2.4 and then simply add up the number of bits in the symbols which make up the code.

The problem, at present, is that the kind of code just mentioned does not take account of gaps in a sequence of hits and the method of correcting for gaps which has been adopted for the time being does not give rise to explicit codes. In Section 4.3 it was suggested that, when the model has been generalised for 'learning', it may be possible to encode gaps by means of explicit codes. If that turns out to be true, then the calculation of $B_E$ can certainly be done by adding up the bits in the symbols which make up the code. Pending these developments, the method which is used for calculating $B_E$ is as follows.

For each new alignment, the value of $B_E$ is:

$$B_E = \sum_{i=2}^{a} E_i - S$$

where $E_i$ is the encoding cost of the Old pattern appearing on one of *a* lines of the alignment other than the top line (where New appears) and S is the saving in encoding costs arising from the fact that some patterns in the alignment convey information about the sequential arrangement of other patterns in the alignment or the selection of other patterns in the alignment where alternatives are possible in a given context.

The encoding cost of any pattern is the value of E for that pattern, calculated as described in Section 5.2. Notice that if any pattern appears two or more times in the alignment, its encoding cost is added a corresponding number of times to the sum of encoding costs.

The calculation of $B_E$ depends on three main ideas:

- As previously noted, a pattern may be fully or partially recognised by any subsequence of the pattern. In other words, it is not necessary to use the specific symbols which were used in calculating the value of E for that pattern. As a general rule when the grammar is largely free of redundancy, if the M values of the relevant symbols (adjusted for gaps - see next) add up to the value of E, then that subsequence of symbols will identify the pattern uniquely amongst the other patterns in the grammar. Where there is redundancy in the grammar, more bits may be needed to achieve unique identification of a pattern.
- If there are gaps in a sequence of hits, information values must be reduced in accordance with the rules used in calculating the value of $B_N$ (Section 5.4)
- For present purposes, there is nothing to be gained by over-specifying a pattern. If one pattern matches a second pattern by the minimum number of symbols to achieve identification of that second pattern, then the saving in encoding costs from this source is maximal. Any additional hits between the two patterns do not give any additional saving in encoding costs.

$B_E$ is calculated in the following way:

1. For each row (R) in the alignment corresponding to a pattern from Old, create a variable (V) containing the value of E for the pattern in that row.
2. Traverse the alignment from left to right examining the columns containing two or more symbols (including symbols in New). Any such column is designated a 'hit' column ($C_H$).
3. For each $C_H$ which contains two or more symbols from patterns in Old (which we may designate $C_{HO}$), examine each row which has a hit symbol from Old in the column (designated $R_{HO}$). For this symbol, calculate $M_A$, an 'adjusted' value of M for the symbol, taking account of any gap which may exist between the given $C_{HO}$ and any previous $C_H$. The method of making the adjustment is the same as is used for calculating the value of $B_N$ (Section 5.4) except that, for each $R_{HO}$, the gaps (or spans) are measured as if all the rows in the alignment *except* the given $R_{HO}$ is

treated as if it were a single pattern to which the pattern in the given $R_{HO}$ is aligned. As in the calculation of $B_N$, it is assumed that there is no gap associated with the first $C_H$ for any given pattern.

4 For each $C_{HO}$, examine each $R_{HO}$ and, amongst these rows, identify the 'leading' row, $R_{HOL}$, whose pattern starts furthest to the left in the alignment (if there is a tie, make an arbitrary choice amongst the ties). For example, in Figure 4, for either of the two columns which contains a hit between '`D`' in '`D 0 t h i s #D`' and '`D`' in '`NP D #D N #N #NP`', the $R_{HOL}$ is the one containing '`NP D #D N #N #NP`' (row 4 in the first case and row 9 in the second case); for either of the two columns containing a hit between '`NP`' in '`NP D #D N #N #NP`' and '`NP`' in '`S NP #NP V #V NP #NP #S`', the $R_{HOL}$ is the row containing '`S NP #NP V #V NP #NP #S`' (row 6 in both cases).

5 For each $C_{HO}$, consider, in turn, each $R_{HO}$, *excluding the* $R_{HOL}$. For each row considered, subtract the value of $M_A$ from the value of V for that row. If the new value of V is less than 0, V is set to 0 and no further subtraction from that instance of V is allowed.

6 When all relevant columns have been examined and the values of the V variables have been reduced, calculate

$$B_E = \sum_{i=2}^{a} V_i$$

where $a$ is the number of rows in the alignment and the summation excludes the top line (which contains New).

The rationale for this method of calculating $B_E$ is that it gives us the sum of the E values of the patterns from Old corresponding to each row of the alignment after the first, with a reduction for hits between those patterns (with an adjustment for gaps as outlined above).

The reason for reducing the value of $B_E$ when there are hits between patterns in Old is that any such hit reflects a degree of 'coverage' of one pattern from Old by another such pattern. To the extent that one pattern provides information that also exists in another pattern there is a reduced need for the second pattern to be identified in the encoding. In the extreme case, where two patterns are identical, only one of them need be identified in the encoding.

As indicated above, any saving in encoding costs resulting from the coverage of one or more patterns by another cannot exceed the E value for each pattern - any additional hits are 'wasted'. Hence, the V value for any row cannot be reduced below 0.

In the method described above, the 'leading' row for any one column ($R_{HOL}$) is regarded as the row with which the other symbols in the column are unified. Hence, for the given column, this is the row where the V value is *not* reduced by the value of $M_A$. Intuitively, the left-to-right bias in the definition of 'leading row' is less theoretically 'clean' than if all concepts were entirely symmetrical between left and right directions in the alignment. However, the concepts as described are the best to have been found so far and seem to work quite well.

## 6 The SP61 model

It is not appropriate to present a detailed description of the SP61 model here since the SP61 model is similar to the SP52 model and a fairly full description of that model may be found in [42]. The main difference between the two models is that SP61 has been generalised to calculate probabilities of inferences. Also, in the development of the model, a fairly large number of minor refinements were made.

At the heart of both models is a process for finding alignments between two patterns which are 'good' in terms of IC. An early version of this method for aligning two patterns is described in [48] and a more refined version in [50].

The process may be regarded as a form of dynamic programming (see, for example, [30]) which differs from standard methods in two main ways:

- It can find alternative alignments between two patterns, graded in terms of compression.
- It exploits list processing techniques so that arbitrarily long patterns may be compared (within the limits of the host machine).
- The thoroughness of searching, and thus the computational resources which are required (time or space or both), may be controlled with parameters.

For each pattern in New (and there is normally only one), this process is applied initially to compare the pattern in New with all the patterns in Old. From amongst the alignments which are found, the best ones are 'unified' to convert the alignment into a single sequence or pattern of symbols. This pattern is stored together with the alignment from which it was derived. Any alignments which cannot be unified in this way are discarded.

For each pattern amongst the best of the unified alignments just formed, the process is repeated, the best alignments are unified to form simple patterns and these patterns (with their alignments) are stored as before. This cycle is repeated in the same way until no more alignments can be found.

The organisation of the model is outlined in Figure 6 using pseudocode. Two aspects of the model, as described in the figure, need some clarification: the nature of the 'hit structure' used in the model and the method by which alignments are selected at the end of each *compress()* cycle.

```
main()
{
    1 Read the rules (patterns) of the knowledge base, each one with
        a frequency of occurrence in an actual or notional sample
        from the World, and store the patterns with their frequencies in Old.
    2 Read the pattern(s) to be parsed and store it (them) in New.
    3 From the frequency of each pattern, derive a frequency for each symbol in
        Old (as described in Section 5.1)
    4 Using the frequencies of the symbols with the method described in the
        text (Section 5.1), assign to each symbol type
        in New and Old a number of bits representing the
        'minimum' information 'cost' of that symbol type. Also, calculate
        an 'actual' information cost for each symbol type (see Section 5.1).
    5 For each pattern in Old, calculate E, the minimum number of bits needed
        to encode that pattern (see Section 5.2).
    7 while (there are more patterns in New to be encoded)
    {
        7.1 Select the first or next pattern in New to be encoded and add it as
            the first 'driving pattern' to an otherwise empty list of
            driving patterns.
        7.2 while (there are more windows in the current pattern)
        {
            7.1.1 Select the first or next 'window' from the
                current pattern from New.
            7.2.2 while (new alignments are being formed)
                compress ()
            7.2.3 Out of all the alignments which have been formed
                for the current pattern up and including the
                current window, print the ones that have been
                selected by compress().
        }
        7.3 Out of all the new alignments which have been formed for
            the entire current pattern from New, print the ones that
            have been selected by compress().
        7.4 Identify a reference set of alignments which encode all and only
            the same symbols from New as the best alignment. Calculate
            absolute and relative probabilities for these alignments and
            for the patterns and symbols in the alignment.
    }
}
```

(continued)

Figure 6 (continued)

```
compress()
{
    1 Clear the 'hit structure' (described in the text).
    2 while (there are driving patterns which have not yet been processed)
    {
        2.1 Select the first or next driving pattern in the set of
            driving patterns.
        2.2 while (there are more symbols in the current driving pattern or
            window of the current pattern from New)
        {
            2.2.1 Working left to right through the current
                driving pattern (or window if the current
                driving pattern is from New), select the first or
                next symbol in the pattern.
            2.2.2 'Broadcast' this symbol to make a yes/no match with
                every symbol in the 'target patterns': the original
                patterns in Old and the alignments in the store of
                alignments formed thus far by the program.
            2.2.3 Record each positive match (hit) in a 'hit structure' (as
                described in the text). As more symbols are
                broadcast, the hit structure builds up a record
                of sequences of hits between the driving pattern
                and the several target patterns in Old and the store
                of alignments. As each hit sequence is extended,
                the compression score of the corresponding
                alignment is estimated using a 'cheap to
                compute' method of estimation.
            2.2.4 If the space allocated for the hit structure is filled at
                any time, the system 'purges' the worst hit
                sequences from the hit structure to release
                more space. The selection uses the estimates
                of compression scores assigned to each
                hit sequence in Step 2.2.3.
        }
    }
```

(continued)

Figure 6 (continuation of the *compress()* function)

3 For each hit sequences which has an estimated compression score above some threshold value and which will 'project' into a single sequence (as described in the text), convert the hit sequence into the corresponding alignment. Discard this alignment if it is identical with any alignment already in Old. Otherwise, compute the compression score using the method described in Section 5, print the new alignment and add it to the store of alignments created by the program. If no new alignments are formed, quit the *compress()* function.

4 Examine all the alignments that have been created since the beginning of processing and choose a 'best' subset of these alignments as described in the text. Remove from the store of alignments all the alignments which have not been selected.

5 Clear the list of driving patterns and then, using the same method as is used in 4 but (usually) with a more restrictive parameter, select a subset of the alignments remaining in the store of alignments and add references to those alignments to the list of driving patterns for the next cycle. (these patterns are not removed from the store of alignments and may therefore also be target patterns on the next cycle).

}

**Figure 6**

An outline, in pseudocode, of the organisation of the SP61 model.

### 6.1 The 'hit structure'

On each application of the *compress()* function shown in Figure 6, a set of alignments is identified, each one between a 'driving' pattern and a 'target' pattern. Since the driving pattern or the target pattern or both may represent an alignment formed at an earlier stage, the model can build alignments with any number of rows.

The *hit structure* is the data structure used to record hits between driving patterns and target patterns. In effect, it is a set of lists where each list records a sequence of hits between one driving pattern and one target pattern. To save space and to facilitate the process of updating the set of lists, a tree structure is used where each path from the root node to a leaf node records a sequence of hits between one driving pattern and one target pattern.

### 6.2 Selection of alignments at the end of each application of the *compress()* function

In Step 4 of the *compress()* function shown in Figure 6, alignments are selected for retention in the store of alignments and the rest are discarded. The process of selection is not simply a matter of choosing the alignments with the highest CD values. This is because it often happens that a majority of the alignments, often with relatively high CD values, all represent encodings of the same symbols from within New and a minority of the alignments, which may have relatively low CD values, encode symbols from elsewhere in New. If selection was to be made purely on the strength of high CD value then these ‘minority’ alignments would be lost. Since they are often stepping stones on the path to the building of alignments which encode all the symbols in New, removing them from the store of alignments may prevent the system ever finding these ‘full’ alignments for New.

The solution to this problem is to assign a ‘quota’ to each of the symbols in New, the same value for all the symbols. Selections are then made for each symbol in New, choosing alignments where the given symbol forms a hit within the alignment and, amongst those alignments, choosing the ones with the highest CDs until the quota is filled or there are no more suitable alignments. Any one alignment may appear in the quota for two or more symbols from New. All alignments which have not been assigned to any quota are discarded.

The overall effect of this method of selection is to ‘protect’ alignments which may have low CD values but which are needed as stepping stones on the way to building alignments which ‘encode’ all or most of the symbols in New and which have higher CD values than would otherwise be obtained.

The same method is used in Step 5 of the *compress()* function to select driving patterns for use in the next cycle. When driving patterns are selected, the quota for each symbol from New is, typically, smaller than when alignments are selected to be retained in the store of alignments. Normally, this means that the driving patterns chosen at the end of each cycle are a subset of the alignments chosen to be retained at the end of each cycle.

### 6.3 Computational complexity

Given that all the example grammars in this article are much smaller than would be required in any realistic system, and given the well-known computational demands of multiple alignment problems, readers may reasonably ask whether the proposed framework for parsing would scale up to handle realistically large knowledge structures and patterns in New.

It has been shown elsewhere [50, 49] that the time complexity of the function (SP21) for finding good partial matches between two patterns, which is embedded in the *compress()* function, is $O(nm)$ in a serial processing environment, where $n$ is the sum of the lengths of the patterns in New and $m$ is the sum of the lengths of the patterns in Old. This big O value is typical of systems which, like SP21, perform dynamic programming or something similar. In a parallel processing environment, the time complexity should approach $O(n)$, $n \leq m$, depending on how the parallel processing is applied. In all environments, the space complexity is $O(m)$ - which is substantially better than the space complexity of any ‘standard’ dynamic programming algorithm.

In SP61, alignments are built by pairwise alignment of patterns and alignments. The number of such pairings required to build an alignment for a whole sentence appears to be approximately $\log_2 (n / c)$, where $n$ is the length of the pattern in New (in symbols) and $c$ is a constant.

The values of $n$ and $m$ for successive pairings will vary, starting relatively small when $n$ is the length of the pattern in New and $m$ is the sum of the lengths of the patterns in the grammar and becoming larger when new alignments have been added to the store and selected as driving patterns. However, the selection process described in Section 6.2 ensures that, after the initial growth, the sizes of $n$ and $m$ remain relatively constant: alignments in the later stages are longer than in the early stages but there are fewer of them. For the purpose of assessing computational complexity, it seems reasonable to assume that variations in the sizes of $n$ and $m$ from the start to finish of processing can be discounted.

Overall, the time complexity of SP61 in a serial processing environment should be approximately $O(\log_2 n \cdot nm)$, where $n$ is the length of the sentence and $m$ is the sum of the lengths of the patterns in Old. In a parallel processing environment, the time complexity may approach $O(\log_2 n \cdot n)$, depending on how the parallel processing is applied. The space complexity should be $O(m)$.

In summary, there is reason to think that the method of forming alignments which is embodied in SP61 will not lead to running times or demands for storage which are hopelessly impractical when the this approach is applied to realistically large examples.

## 7 Probabilistic reasoning, multiple alignment and information compression

What connection is there between the formation of an alignment, as described in Section 2.1.1, and PrbRs? Using simple examples, this section tries to show informally how PrbRs may be understood as the formation of an alignment in which some part or parts of the patterns from Old which appear in the alignment (single symbols, substrings or subsequences within patterns from Old) are not aligned with any matching symbol or symbols from New. To the extent that the alignment is a ‘good’ alignment, the unmatched symbols from Old which appear within the alignment may be interpreted as probabilistic inferences. As a working hypothesis, all kinds of PrbRs may be understood in these terms.

What connection is there between the probability of any inferences which may appear in an alignment and measures of compression for that alignment? How can the probability of MA inferences be calculated? Answers to these two questions are presented at the beginning of the second of these three articles [40].

### 7.1 Scene analysis and pattern recognition

The idea that pattern recognition might be understood as IC has been recognised from at least as far back as 1972 ([38]). At first sight, pattern recognition and IC seem unrelated and, at first sight, it is not obvious what either of them has to do with PrbRs. However, a little reflection should make clear how these things are related.

Consider how we perceive a typical scene. Outdoors, for example, we may see a tree, behind which is parked a car which is itself in front of a house. With no conscious effort we immediately recognise the tree, the car and the house despite the fact that part of the car is obscured by the tree and part of the house is obscured by the tree and the car.

Not only can we recognise a whole object or pattern from a subset of its parts but, within wide limits, it normally does not matter very much which parts we see and which are obscured. This process of recognising objects in a scene using partial information, which is so familiar and so apparently effortless, may be interpreted as a form of MA. It may also be interpreted as a form of PrbRs.

Figure 7 shows a an MA which represents, schematically, a side view of how the scene described above is recognised.

```
h h h h h c c c t t t c c c c h h h h h
| | | | | | | | | | | | | | | | | | | |
| | | | | | | | t t t | | | | | | | | |
| | | | | | | |       | | | | | | | | |
| | | | | c c c c c c c c c c | | | | |
| | | | |                     | | | | |
h h h h h h h h h h h h h h h h h h h h
```

**Figure 7**

An alignment illustrating schematically the recognition of a house, car and tree. The top row represents a side view or cross-section of the scene as it is projected onto the retina of one eye. The second row (‘`t`’) represents a side view or cross-section of a record in memory for the appearance of a tree. Rows three and four represent similar records for a car (‘`c`’) and a house (‘`h`’).

The symbols ‘`t`’, ‘`c`’ and ‘`h`’ represent elements of the tree, car and house, respectively. The top row represents a side view or cross-section of an image of the scene as it appears on the retina of one eye. The three rows below represent similar views of three stored patterns corresponding to the three types of object.

There is, of course, much more to the recognition process than is conveyed by the figure. But, for recognition to occur, there must be a mapping between elements of the image and elements of the stored patterns. The alignment shown in Figure 7 seems to capture the overall structure of that mapping.

Where does reasoning figure in this kind of partial pattern recognition? By recognising the car and the house we infer the existence of the parts which we cannot see but which appear in our memories of typical cars and houses, or our memory of a specific house or car. We may, for example, infer that the handle of one door of the car is hidden behind the tree. Likewise, we may infer that unseen parts of the house are composed of brick or stone in accordance with our memory of what houses are built from or what one specific house is built from. If we can see just the top half of a door, we are likely to infer that we would see the bottom half if the car were to be moved or if we looked behind the car.

The inferences are probabilistic because none of them are certain. In some unlikely world, there might simply be gaps in the car and the house at the precise points where those gaps are covered by the object in front. Or the part of the house which we cannot see may be made of paper, not brick or stone - or it could conceivably be made of green cheese! Whether something exists when it is unseen (or otherwise perceived) is, of course, a long-standing issue in philosophy.

## 7.2 Reasoning with rules

The kind of partial pattern recognition discussed in the previous subsection is, perhaps, not normally regarded as a paradigm for PrbRs. This subsection tries to show informally, by means of a simple example, that reasoning with ‘rules’ - which is, perhaps, closer to what most people understand as ‘reasoning’ - might also be understood in terms of ICMAUS. In this context, a ‘rule’ is something which expresses a relationship between a ‘condition’ and an associated ‘action’ or ‘consequence’ - like a rule in an expert system.

We are all familiar with the dictum “There is no smoke without fire”. Although this is often used in a metaphorical sense, it expresses, in a slightly inaccurate form, our experience of real smoke and real fires: that (almost) every fire produces smoke and that (almost) all smoke is generated by a fire.

A simple and natural way to represent this association, ‘rule of thumb’ or regularity about the world is by means of a simple pattern containing a symbol for smoke and a symbol for fire, something like this: ‘`fire smoke`’ or, perhaps, ‘`smoke fire`’.

To honour fully the principle that knowledge structures should reflect the ‘raw’ data from which they are derived (see Section 1.3), representation of the association between smoke and fire would need three spatial dimensions and one temporal dimension. Since we have chosen, for the time being, to confine ourselves to one-dimensional patterns, the full structure of the association between smoke and fire can be represented only approximately. In the example patterns, above, the first version is, possibly, better than the second because quantities of smoke which can be seen easily first appear after the fire which creates it has started. However, for many applications, this kind of detail is not important and the basic fact that smoke and fire tend to occur together is what matters.

If we see some smoke, this new observation may be represented in ‘New’ with a pattern containing a single symbol for smoke, thus: ‘`smoke`’. This pattern may be matched with ‘Old’ knowledge which includes the pattern ‘`fire smoke`’.

Amongst the ‘good’ alignments arising from this matching process we are likely to find this alignment:

```
      smoke
        |
fire  smoke
```

As in the previous example of partial pattern matching (Section 7.1), any unmatched symbol in the alignment may be interpreted as an inference which the system has made. In this case, the existence of smoke has led to an inference that there is likely also to be a fire in the same vicinity. In most cases, the inference will not be certain because ‘`smoke`’ may also align with patterns relating to artificial smoke used in dramatic productions where there is no associated fire.

This subsection has considered an extremely simple example of reasoning with a rule. More elaborate examples are presented in [40] and [41], including examples of ‘chains’ of reasoning in two or more steps.

## 8 Conclusion

This, the first of three articles introducing the idea that PrbRs may be understood as ICMAUS, has introduced the concept of multiple alignment as it has been developed in this work and has explained how IC associated with any such alignment may be calcluated. The SP61 model for the formation of alignemnts has been described in outline and, in outline, it has been shown how probabilistic inferences may be derived from alignments.

The next article [40] starts with a description of how probabilities of inferences may be calculated. The remainder of [40] and most of the third article [41] present a range of examples to show how different aspects and kinds of PrbRs may be understood in these terms.

## Acknowledgements

I am grateful to Prof. C. S Wallace of Monash University, to Dr. Tim Porter of the Department of Mathematics, University of Wales, Bangor (UWB) and to Mr. John Hornsby of the School of Electronic Engineering and Computer Systems, UWB, for useful discussions of various ideas presented in this article and the two articles which accompany it.

## References

1 Allison, L and Wallace, C. S. (1994) The posterior probability distribution of alignments and its application to parameter estimation of evolutionary trees and to optimization of multiple alignments. *Journal of Molecular Evol*ution 39, 418-430.

2 Allison, L, Wallace, C. S. and Yee, C. N. (1992) Finite-state models in the alignment of macromolecules. *Journal of Molecular Evol*ution, 35, 77-89.

3 Barton G. J. (1990) Protein Multiple Sequence Alignment and Flexible Pattern Matching. *Methods in Enzymology,* 183, 403-428.

4 Belloti, T. and Gammerman, A. (1996) Experiments in solving analogy problems using Minimal Length Encoding. Presented at Applied Decision Technologies '95, Brunel University, April 1995 (Proceedings of Stream 1, *Computational Learning and Probabilistic Reasoning*, pp. 209-220).

5 Bondarenko, A., Dung, P. M., Kowalski, R. A. and Toni, F. (1997) An abstract, argumentation-theoretic approach to default reasoning. *Artificial Intelligence* 93(1-2), 63-101.

6 Chan, S. C., Wong, A. K. C. and Chiu, D. K. Y. A (1992) Survey of Multiple Sequence Comparison Methods. *Bulletin of Mathematical Biology*, 54 (4), 563-598.

7 Cheeseman, P. (1990) On finding the most probable model. In J. Strager and P. Langley (Eds.) *Computational models of scientific discovery and theory formation*, Chapter 3, Morgan Kaufmann, San Mateo, California, pp.73-95.

8 Cover, T. M. and Thomas, J. A. (1991). *Elements of Information Theory*. New York: John Wiley.

9 Cussens, J. and Hunter, A. (1992) Using maximum entropy in a defeasible logic with probabilistic semantics. In *Proceedings of the International Conference in Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU 92)* and in *Lecture Notes in Computer Science*, vol. 682, pp. 43-52, Springer.

10 Dagun, P. and Luby, M. (1997) An optimal approximation algorithm for Bayesian inference. *Artificial Intelligence* 93(1-2), 1-27.

11 Day, W. H. E. and McMorris, F. R. (1992) Critical Comparison of Consensus Methods for Molecular Sequences. *Nucleic Acids Research,* 20 (*5*), 1093-1099.

12 Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of Molecular Evolution,* 17, 368-376.

13 Gabbay, D. M., Hogger, C. J. and Robinson, J. A. (Eds.) (1994) *Handbook of Logic in Artificial Intelligence and Logic Programming, Volume 3, Nonmonotonic Reasoning and Uncertain Reasoning*. Oxford: Clarendon Press.

14 Gammerman, A. J. (Ed.) (1996). *Computational Learning and Probabilistic Reasoning*. Chichester: John Wiley.

15 Gammerman, A. J. *(*1991) The representation and manipulation of the algorithmic probability measure for problem solving. *Annals of Mathematics and Artificial Intelligence,* 4, 281-300.

16 Ginsberg, M. L. (1994) AI and non-monotonic reasoning. In D. M. Gabbay, C. H. Hogger and J. A. Robinson (Eds.), *Handbook of Logic in Artificial Intelligence and Logic Programming, Vol. 3, Non-monotonic Reasoning and Uncertain Reasoning*. Oxford: Clarendon Press, pp. 1-33.

17 Grünwald, P. (1997) The Minimum Description Length Principle and Non-Deductive Inference. *Proceedings of the IJCAI Workshop on Abduction and Induction in AI* (editor, P. Flach), Nagoya, Japan.

18 Grünwald, P. (1998) Minimum encoding approaches for predictive modelling. To appear in the *Proceedings of the Fourteenth International Conference on Uncertainty in Artificial Intelligence* (Madison, WI, 1998).

19 Hortvitz, E. J., Breese, J. S. and Henrion, M. (1988) Decision theory in expert systems and artificial intelligence. *International Journal of Approximate Reasoning 2*, 247-302.

20 Kern-Isberner, G. (1998) Characterising the principle of minimum cross-entropy within a conditional-logical framework. Artificial Intelligence 98(1-2), 169-208.

21 Kohlas, J., Haenni, R. and Mouney, P. A. (1998) Model-based diagnostics and probabilistic assumption-based reasoning. *Artificial Intelligence* 104(1-2), 71-106.

22 Li, M. and Vitányi, P. *(*1997) *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, New York.

23 Lowry, R. (1989). *The Architecture of Chance*. Oxford: Oxford University Press.

24 Parsons, S. and Hunter, A. (1998). A review of uncertainty handling formalisms. In "Applications of Uncertainty Formalisms", *Lecture Notes in Computer Science*, vol. 1455.

25 Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Francisco, California.

26 Pednault, E. P. D. (1991) Minimal-length encoding and inductive inference. In G. Piatetsky-Shapiro and W. J. Frawley (eds.), *Knowledge Discovery in*

*Databases*, MIT Press, Cambridge Mass.

27 Reichert, T. A., Cohen, D. N. and Wong, A. K. C. (1973) An application of information theory to genetic mutations and the matching of polypeptide sequences. *Journal of Theoretical Biology,* 42, 245-261.

28 Rissanen, J. (1978) Modelling by the shortest data description. *Automatica-J,.*IFAC 14, 465-471.

29 Roytberg, M. A. A. (1992*)* Search for Common Patterns in Many Sequences,. *Cabios,* 8 (1), 57-64.

30 Sankoff, D. and Kruskall, J. B. (1983) *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparisons*. Reading, MA: Addison-Wesley.

31 Schaub, T. and Bruning, S. (1998) Prolog technology for default reasoning: proof theory and compilation techniques. *Artificial Intelligence* 106(1), 1-75.

32 Schurz, G. (1998) Probabilistic semantics for Degrande's conditional logic and a counter example to his default logic. *Artificial Intelligence* 102(1), 81-95.

33 Solomonoff, R. J. (1964). A formal theory of inductive inference. Parts I and II. *Information and Control* 7: 1-22 and 224-254.

34 Storer, J. A. *(*1988) *Data Compression: Methods and Theory*. Computer Science Press, Rockville, Maryland.

35 Taylor, W. R. (1988) Pattern matching methods in protein sequence comparison and structure prediction. *Protein Engineering,* 2 (2), 77-86.

36 van der Gaag, L. C. (1996) Bayesian belief networks: odds and ends. *Computer Journal 39*, 97-113.

37 Wallace, C. S. and Boulton, D. M. (1968) An information measure for classification. *Computer Journal,* 11 (2), 185-195.

38 Watanabe, S. (1972) Pattern recognition as information compression. In S. Watanabe (Ed.), *Frontiers of Pattern Recognition*, New York: Academic Press.

39 Wolff, J. G. (submitted, a) Probabilistic reasoning as information compression by multiple alignment, unification and search (I): introduction (This article).

40 Wolff, J. G. (submitted, b) Probabilistic reasoning as ICMAUS (II): calculation of probabilities, best-match pattern recognition and information retrieval, and reasoning with networks, trees and rules.

41 Wolff, J. G. (submitted, c) Probabilistic reasoning as ICMAUS (III): hypothetical reasoning, geometric analogies, default values, nonmonotonic reasoning, and modelling 'explaining away'.

42 Wolff, J. G. (submitted, d) Parsing as information compression by multiple alignment, unification and search. Two articles on which this paper is based ("Parsing ... SP52" and "Parsing ... Examples") may be obtained from http://www.sees.bangor.ac.uk/~gerry/sp_summary.html.

43 Wolff, J. G. (1997) Causality, statistical learning and multiple alignment. Paper presented at the UNICOM Seminar and Tutorial on Causal Models and Statistical Learning, London, March 1997.

44 Wolff, J. G. (1996a) Logic as information compression by multiple alignment, unification and search. SEECS Report, October 1996. A copy may be obtained from http://www.sees.bangor.ac.uk/~gerry/sp_summary.html.

45 Wolff, J. G. (1996b) Information compression by multiple alignment, unification and search as a universal theory of computing. SEECS Report, February 1996. A copy may be obtained from http://www.sees.bangor.ac.uk/~gerry/sp_summary.html.

46 Wolff, J. G. (1996c) Learning and reasoning as information compression by multiple alignment, unification and search. In: A. Gammerman (ed.), *Computational Learning and Probabilistic Reasoning*, Wiley, Chichester, pp. 67-83. An earlier version was presented at Applied Decision Technologies '95, Brunel University, April 1995 (Proceedings of Stream 1, *Computational Learning and Probabilistic Reasoning*, pp. 223-236).

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

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

49 Wolff, J. G. (1995c) Multiple alignment and computing. SEECS Report, August 1995. A copy may be obtained from http://www.sees.bangor.ac.uk/~gerry/sp_summary.html.

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

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

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

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

54 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.* Lawrence Erlbaum, Hillsdale, NJ. Reprinted in Chapter 2 of Wolff (1991).

55 Wolff, J. G. (1982) Language acquisition, data compression and generalization. *Language & Communication*, 2, 57-89. Reprinted in Chapter 3 of Wolff (1991).

# A1 APPENDIX: DEFINITIONS OF TERMS

### Definition 1

A **symbol** is some kind of mark which can be discriminated from other symbols. In the context of pattern matching, a symbol is the smallest unit which can participate in matching: a symbol can be compared (matched) only with another single symbol and the result of matching is either that the two symbols are the same or that they are different. No other result is permitted.

An important feature of the concept of a *symbol*, as it is used in this research, is that, with respect to the way it behaves within the system, it has ***no hidden meaning***. In this research, a symbol is *nothing but* a primitive mark which can be discriminated in a yes/no manner from other symbols. There are no symbols like the symbols in an arithmetic function (e.g., '6', '22', '+', '-', '×', '/', '(', ')' etc), each of which has a meaning in the system which is not directly visible.[18]

Any symbol in the present scheme may have a meanings but it must take the form of one or more other symbols which are associated with the given symbol and are explicit and visible within the structure of symbols and patterns.

Although symbols in the proposed framework have no hidden meaning having an effect on how the symbol behaves in the system, they can and often do have a meaning for the human reader.

In SP6, a symbol is any string of characters (not including the space character) which is bounded at each end by spaces.

**Related concepts**

If two symbols match, we say that they belong to the same **symbol type**. In any system which contains symbols, we normally recognise an **alphabet** of symbol types such that every symbol in the system belongs in one and only one of the symbol types in the alphabet, and every symbol type is represented at least once in the system.

A positive match between two symbols is termed a **hit**. One or more unmatched symbols within one pattern between two hits is a **gap**.

**Definition 2**

A **pattern** is an array of symbols in one, two or more dimensions. For present purposes, one dimensional patterns (**sequences** or **strings** of symbols) will be the focus of attention.

*The meaning of the term pattern includes the meanings of the terms substring and subsequence (Definition 3 and Definition 4, below).*

**Comment**

Notwithstanding the definition of *symbol*, above, the start and end of every pattern is marked for the system by a character whose sole purpose is to mark the left and right boundaries of patterns. These characters - which may be left and right brackets ('(' and ')'), single quotes or non-printing end-of-line characters - are not 'symbols' in the sense defined above. They are 'meta symbols', which are necessary for any system like the SP61 model, if New and Old are to be divided into discrete patterns.

**Definition 3**

A **substring** is a sequence of symbols of length $n$ within a sequence of length $m$, where $n \leq m$ and where the constituent symbols in the substring are contiguous within the

18. But see the comment with the definition of *pattern*, below.

sequence which contains the substring.

**Definition 4**

A **subsequence** is a sequence of symbols of length $n$ within a sequence of length $m$, where $n \leq m$ and where the constituent symbols in the subsequence may not be contiguous within the sequence which contains the subsequence. The set of all subsequences of a given sequence includes all the substrings of that sequence.