LOGIC AS INFORMATION COMPRESSION BY MULTIPLE ALIGNMENT, UNIFICATION AND SEARCH

J Gerard Wolff

October 1996

[Running title: “Logic and Multiple Alignment”]

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

Abstract

This article considers the proposal that logic may usefully be understood as information compression by *multiple alignment*, with ‘unification’ and ‘search’, where ‘multiple alignment’ has a meaning which is close to its meaning in bio-informatics, ‘unification’ means a simple merging of matching patterns, and ‘search’ has its normal meaning in the context of computing. A motivation for developing these ideas, not considered in this article, is the possible integration of concepts in logic with other concepts in computing including learning, pattern recognition, information retrieval and others.

A theme throughout this article is that these concepts may model ‘standard’ forms of logic exemplified by propositional logic, first order logic and related ideas but they also offer the possibility of some rationalisation and simplification of concepts in logic.

The way in which basic concepts in logic (AND, OR, NOT etc.) may be modelled in the proposed framework is discussed and some possible rationalisations are proposed.

The way in which the multiple alignment framework may accommodate the composition of complex forms of logic from simpler constituents is described.

Logical deduction is considered both in standard forms and in proposed revisions.

Finally, there is an outline description of how the multiple alignment framework may be applied to a selection of problems associated with nonmonotonic and uncertain reasoning: inheritance and default reasoning (including the negation of a default attribute), abductive reasoning and inductive reasoning.

1. INTRODUCTION

This article considers the proposal that logic may usefully be understood as information compression and, more specifically, that logic may usefully be understood a process of *multiple alignment*, with 'unification' and 'search', where 'multiple alignment' has a meaning which is close to its meaning in bio-informatics, 'unification' means a simple merging of matching patterns,[1] and 'search' has its normal meaning in the context of computing.

That logic may have any significant relation to information compression (IC) may, at first sight, seem an outlandish idea. But consider an elementary form of 'deduction' which is illustrated by how we perceive the scene shown in Fig. 1. We interpret this scene as a building with a car in front of it and with a tree in front of the car and the building. Although some parts of the building are obscured by the car and the tree, and some parts of the car are obscured by the tree, we naturally assume that the unseen parts are 'really' present.

Please insert Fig. 1 about here

Fig. 1. A scene in which some objects are partially obscured by other objects.

This process of inference of the unseen parts of a scene from those parts which have been seen - which is so prominent in our everyday perceptions of the world - may be viewed as a primitive form of 'deduction'. From the parts of each object or pattern which we can see, and from our knowledge of what the whole object or pattern looks like, we deduce what parts are hidden from view.

Making the connection between patterns of information on the retina and patterns stored in our brains is a process of recognition. It has been understood for some time that recognition can be understood as a process of IC (Watanabe, 1972, 1985). Thus inference or 'deduction' which depends on the recognition of a whole pattern from a subset of its parts may also be understood as IC.

It is not hard to see how recognition may be understood as IC: we suppose that the external information in the patterns which are fully or partly seen is merged ('unified') with the matching patterns which we have already stored in our brains; in each case where an external pattern is unified with a stored pattern, there is an overall compression of information because two patterns are reduced to one. If an external pattern can be successfully unified with a stored pattern, we may communicate an occurrence of the pattern using a short identifier like 'building', 'car' or 'tree', in essentially the same way that relatively short 'codes' are used to identify the occurrence of

1. In logic, 'unification' means a process of making substitutions for variables in each of two formulae so that the two formulae become identical and may thus (implicitly or explicitly) be merged to form a single instance. The meaning of 'unification' in the present context is the simpler but obviously related meaning of merging two identical patterns into one. Throughout this article, the term will normally be used with its simpler meaning. Where the 'logic' meaning of the term is intended, this will be clearly marked.

relatively large patterns in standard methods for IC (see, for example, Storer (1988)). Another example of the use of a short code to represent a larger pattern is the use of ‘IC’ in this article to represent the phrase ‘information compression’.

In case the example of scene analysis, just described, seems too remote from logic in its traditional sense, consider the following simple syllogism:

> *If today is Tuesday, then tomorrow will be Wednesday.*
> *Today is Tuesday.*
> *Therefore, tomorrow will be Wednesday.*

The fact that Wednesday always follows Tuesday is a familiar pattern of events described by the first proposition. The observation that today is Tuesday leads us to the conclusion that tomorrow will be Wednesday because we can unify this observation with the larger pattern represented by the first proposition. (No doubt this larger pattern is itself part of an even larger pattern that we learned as children listing all seven days of the week in their correct order).

The concept of ‘unification’ as a simple merging of identical patterns is simpler than the concept of ‘unification’ in logic but it is a prominent part of that concept. Given that ‘unification’ in the sense of logic is a key concept in that field, and given that it embraces a concept with a clear connection with IC, there is further support for the idea that there may be a more than superficial relation between logic and IC.

1.1. Multiple alignment, unification and search

In the rest of this article, the phrase ‘information compression by multiple alignment, unification and search’ will normally be abbreviated as ‘ICMAUS’.

As described later in this article, the ICMAUS framework may be capable of modelling a range of established concepts in propositional logic, first order logic and related areas. If this were all that it could do, readers might reasonably ask “Has anything been gained by representing existing concepts in a new form?” At least two possible justifications for developing these proposals may be offered:

- ICMAUS may facilitate the integration of concepts in logic with other aspects of computing such as learning (Wolff, 1996), parsing (Wolff, submitted, b), best-match information retrieval (Wolff, 1994), fuzzy pattern recognition, and others. The possible application of ICMAUS to aspects of computing other than logic has been considered in the articles cited, in Wolff (submitted, a) and elsewhere.[2]

2. The application of ICMAUS concepts to aspects of computing other than logic are discussed in articles, some of which are not yet published but which are available from the author on request. It would take too much space and take us too far afield to attempt to summarise this work here. As far as the author is aware, there are no other articles on this subject by other authors which can be cited at the time of writing.

- Within the proposed framework, there seems to be some scope for rationalising and simplifying concepts in logic and integrating different areas within the field.

The possibility of rationalising some aspects of logic will be a recurring theme throughout this article which will be considered alongside discussion of possible ways in which established concepts may be modelled as ICMAUS. For brevity, ‘LS’ is used as a shorthand for ‘standard’, established concepts in propositional logic, first order logic and related areas (as described in Copi (1986), Klop (1992) and Davis (1993)). ‘LR’ is used to mark concepts which are, in some sense, ‘rationalised’ or ‘revised’.

Although ICMAUS provides a framework which may accommodate LR concepts, some of these ideas are similar to ideas which are already well-established and can be seen, for example, in Prolog. However, it should be emphasised that the ICMAUS proposals are not merely syntactic variations on Prolog - they extend into the realms of fuzzy and probabilistic matching, well beyond the clockwork operations of Prolog. The LR proposals outlined in Section 7 appear to be radically new.

1.2. The status of these proposals

These proposals are by no means a polished and ‘complete’ set of ideas. They are the result of a fairly long period of study and reflection, driven by a strong intuition that this line of thinking offers potential dividends if answers can be found to the many questions and problems which naturally arise in any attempt to develop a new conceptual framework which will accommodate a wide range of established concepts, including concepts in logic.

A computer model is under development which incorporates several of the ideas described in this article. A description of the model and its application to parsing is presented in Wolff (submitted, b). Some of the thinking in the current model is outlined briefly in Section 3.1 but a fuller description of the model would be inappropriate in this article because the model is not yet sufficiently mature to demonstrate all the concepts described below.

I believe that the overall framework of ideas is already sufficiently clear and sufficiently novel to be of interest to readers now. I hope that readers will carry the thinking further, either to expose weaknesses which have not yet been seen or to develop and refine the ideas into a more robust form.

1.3. Presentation

In what follows, Section 2 presents some preliminary ideas as background and context to what follows. Section 3 describes what ‘multiple alignment’ means in this programme of research and describes in outline how it relates to IC, probabilities and search. Section 4 describes how some basic concepts in logic may be modelled in the ICMAUS framework. Section 5 describes how, in this framework, simpler structures in logic may be assembled or ‘composed’ to make more complex structure. Section 6 considers how deductive reasoning may be modelled as ICMAUS. And Section 7 examines the possibility that nonmonotonic and uncertain reasoning may also be understood in these terms.

2. PRELIMINARIES

2.1. Background

The proposals in this article have their origins in three main strands of thinking:

- From at least as long ago as the writings of Ernst Mach in the last century there has been recognition of a close connection between human thinking and information compression (see, for example, Zipf (1949), Attneave (1954), Von Békésy (1967), Barlow (1969), Wolff (1991))

- Since Solomonoff's two seminal papers on inductive inference (1964) there has been a stream of research developing the idea that inductive inference and some kinds of inductive problem solving are closely related to information compression (see, for example, Wallace and Boulton (1968), Cook and Rosenfeld (1976), Wallace and Freeman (1987), Rissanen (1987), Wolff (1991), Gammerman (1991)).

- Concepts of inductive inference are closely related to concepts in Algorithmic Information Theory (see, for example, Li. and Vitányi (1993)), based on the idea that randomness in information can be defined as incompressibility of information. Solomonoff's original work is regarded as formally equivalent to AIT.

Some of this thinking is reviewed in Wolff (1993, 1995a).

More immediately, these proposals originate in a programme of research developing the 'SP' conjecture that *all kinds of computing and formal reasoning may usefully be understood as information compression by pattern matching, unification and search* (Wolff 1990-96). The background and motivation for the research is described most fully in Wolff (1993).

The main difference between the SP programme and other work related to inductive inference or AIT are:

- The SP programme seeks to integrate *all* kinds of computing and formal reasoning within a framework of information compression. The goal is broader than understanding 'inductive inference', unless that term is taken to mean 'all kinds of computing and formal reasoning'.

- The SP programme is based on the hypothesis that *all* kinds of compression may be understood in terms of pattern matching, unification and search. All existing and projected SP models are restricted to pattern matching, unification and search and avoid 'arithmetic coding' and related techniques which are used in some other research related to IC. The restriction has been imposed in the interests of simplicity in the SP theory. If, as conjectured, arithmetic may be understood in terms of pattern matching, unification and search, then arithmetic coding may also be understood in those terms. But until this has been demonstrated, the adoption of arithmetic coding or any similar technique would add unwanted complexity to the SP model.

2.2. Representation of knowledge

A working hypothesis in this programme of research is that, with the aid of established principles for IC, it should be possible to model any kind of knowledge (including any kind of language or notational system) with patterns[3] of atomic symbols in one or more dimensions, *where there are no 'meta' symbols with special meanings* (for further discussion, see Wolff (1995a), Section 3). For the time being, the focus of attention is one-dimensional patterns but, at some stage, the ideas may be generalised to patterns in two or more dimensions.

This hypothesis is motivated in part by the thought that any good theory of knowledge should acknowledge the fact that much of our empirical knowledge is derived from 'raw' data received by our senses and should describe how such raw data may become 'knowledge'. There is good empirical evidence, some of which is reviewed in Wolff (1993), that the path from data to knowledge is a process of information compression. Given a 'principle of least effort', an implications of this view of data, knowledge and learning is that the forms of stored knowledge are likely to have recognisable similarities to the patterns of raw data which are received by the senses.

As emphasised in the first paragraph above, a key part of the hypothesis is that there should be no 'meta' symbols with hidden meanings. By hypothesis, all symbols should enter into matching and unification in the same way and all meanings should derive from the processes of pattern matching, unification and search. Given this 'principle of transparency', one of the tasks of the research is to see whether and how, within the framework which is to be described, it may be possible to model the semantics of meta symbols as they appear in established formalisms. Examples are presented below.

Since the aim is to analyse established concepts into more primitive elements, it should not be surprising if the analysis sometimes appears more complex than what is being analysed. For example, a fragment of logic like '$\neg p$' appears much less simple in terms of ICMAUS (see Section 5.2, below). The suggestion here is that this is a matter of appearance, not reality, and that the ICMAUS analysis has not added anything which was not already implicit in the original representation.

The significance of IC in the hypothesis is that, within the scope of established principles of IC, it ensures that *any* kind of knowledge can be represented in a succinct way. It is a curious fact that although much discussion of how knowledge can or should be represented (in, for example, theoretical linguistics) places emphasis on the idea that knowledge should be represented succinctly, and although many of the techniques which are used for the representation of knowledge can be seen as examples of techniques for IC, the connection between the two areas of research is rarely acknowledged.

---

3. The term *pattern* in this article means any array of symbols in one or more dimensions including a sequence, string, substring or subsequence of symbols. The term *substring* means a sequence of symbols within a larger sequence where the symbols are contiguous within the larger sequence. The term *subsequence* means a sequence of symbols within a larger sequence where the symbols are not necessarily contiguous within the larger sequence.

2.2.1. Representation of atomic symbols

For the sake of clarity and readability of the examples presented below, the convention has been adopted that an atomic symbol is any string of one or more characters bounded by spaces. Although this convention means that the space character is treated as a meta symbol, it should be clear that this is purely a cosmetic matter and that any set of patterns in this form may be translated into a form which does not depend on a special character to separate one atomic symbol from the next.

2.3. Logic and uncertainty

As described below, the ICMAUS framework in its full generality is one in which probabilities are the rule and where it is normally impossible to prove that the best possible result has been found. A working hypothesis is that the all-or-nothing character of traditional kinds of logic may be modelled with probabilities which are close to 0 or 1.

A probabilistic framework has the attraction that it suggests the possibility of developing a unified treatment for both traditional kinds of logic and concepts of nonmonotonic and uncertain reasoning (discussed in Section 7). It also sits well with some other aspects of computing which we may seek to integrate with logic such as unsupervised learning (Wolff, 1996), parsing (Wolff, submitted, b), best-match information retrieval (Wolff, 1994), fuzzy pattern recognition, and others (as previously noted, these matters are considered in the articles cited and elsewhere).

This ‘probabilistic’ approach to logic differs from ‘fuzzy logic’ (see, for example, Klir and Yuan (1995), Kosko (1994)) because it is not a ‘fuzzy’ superset of logic. It seeks to develop a single conceptual framework within which deductive and probabilistic kinds of inference may be integrated and simplified. There are many other differences between the concepts to be described and concepts in the area known as fuzzy logic.

3. MULTIPLE ALIGNMENT, COMPRESSION, PROBABILITIES AND SEARCH

The term ‘multiple alignment’ is borrowed from the field of bio-informatics where it means the arrangement of (symbolic representations of) two or more DNA sequences or two or more sequences of amino acid residues so that matching symbols are aligned vertically in columns. In biochemistry, this kind of analysis is used in the process of determining the structure, function or evolution of the corresponding molecules.

In this area of research on multiple alignment, it is generally recognised that there is normally an astronomically large number of alternative ways in which symbols may be aligned and that some are better than others. This raises two questions: what one might mean by a ‘good’ alignment and how to find ‘good’ alignments amongst the many possibilities. These two questions are considered briefly in the next two sub-sections.

3.1. Compression and probabilities

Intuitively, a ‘good’ alignment is one with many positive matches (‘hits’) between symbols, relatively few ‘gaps’ between hits (where there are unmatched symbols) and any gaps should be as short as possible. In these intuitive terms, the best alignment of a pair of strings with a low

'Hamming distance' is better than the best alignment of a pair of strings with a high Hamming distance.

Since compression may be achieved by unification of matching patterns, it should be clear in broad terms that a 'good' alignment (with many hits and few unmatched symbols) is one in which there is a relatively large scope for IC. In the extreme case, two strings of symbols are identical and may be aligned without any gaps between hits. In this case, the two strings can be reduced to one without loss of non-redundant information (provided a record is kept of the fact that there were originally two copies of the information).

In other cases, there may be only substrings that match. In cases like this, one of a pair of matching substrings is given a 'tag' or 'code', the other substring is deleted and its position is marked with a copy of the tag or code. As previously noted, the use of 'IC' as a shorthand for the phrase 'information compression' is an example of this technique. Of course, it only makes sense to make this substitution if the substrings are sufficiently large to yield a net saving after the cost of the tags is counted.

3.1.1. Probabilities

The connection between concepts of information and probability can be seen in Shannon's classic formula for the average information provided by the occurrence of an 'event':

$$H = \sum^{n} p_i \log p_i$$

where p_i is the probability of the ith event in a set of n alternative events. More simply, the information (in bits) provided by the occurrence of any specific event is:

$$h_i = \log_2 \frac{1}{p_i}$$

In general, the occurrence of an event which is probable conveys less information than the occurrence of an event which is rare.

For a sequence of events, $e_1 \ldots e_m$ (e.g., the characters in an ordinary text), the information conveyed by the sequence is:

$$H_s = \sum^{m} h_j = \log_2 (1 / \prod^{m} p_j)$$

One can either compute the number of bits for each of the several events (using the absolute probability of the event in each case) and then add them up or one can compute a probability for the whole sequence (normally, a very small number) and derive the number of bits from that. The former is normally more convenient than the latter.

An approximation to these values can be obtained if each event is coded in accordance with the Huffman principle (1952) so that commonly occurring events receive short codes and rare events receive long codes. The use of some kind of 'prefix' coding scheme (e.g., 0, 01, 011, 0111, 01111

...) ensures that codes can be discriminated unambiguously even when they are concatonated without breaks between successive codes.

In any body of information in which there are relatively large repeating patterns or relatively frequent repeating patterns, or both, the above formulae are inaccurate because they do not take account of the redundancy represented by the repeating patterns. However, the Huffman principle can also be applied to the tags or codes mentioned earlier used as markers for repeating substrings. Substrings which repeat frequently (and thus have a high probability) should be given shorter codes than substrings which occur rarely. In this way, the redundancy represented by the repeated substrings can be coded and the size of a compressed body of information can be minimised.

Thus, compression as well as information is closely related to concepts of probability: an alignment which gives good compression is also one with a relatively high probability. For further discussion see Allison and Yee (1990), Allison, Wallace *et al.* (1992), Chan, Wong *et al.* (1992), and Allison and Wallace (1994).

3.1.2. Towards a new method for evaluation of alignments

A fuller development of the ideas presented in this article will require precise methods for calculating the compressions and probabilities that may be achieved in the proposed framework. Methods already exist for evaluating alignments in these terms (see the articles cited) but those which have been examined are not fully satisfactory in terms of the principles which have been adopted as working hypotheses in this programme of research. In particular, it seems that none of the existing methods can be applied directly to the generalised version of the multiple alignment problem described in Section 3.3 and none of them stay strictly within the framework of pattern matching, unification and search.

Steps towards a new method are described in Wolff (submitted, b). In summary, this new method treats the alignment problem as one of finding an economical encoding for a sequence of symbols designated as New in terms of other sequences designated as Old. Each sequence in Old has an associated cost which is the number of bits required to discriminate the beginning and end of the sequence together with the number of bits required to identify the sequence uniquely amongst the set of Old sequences. The ‘compression score’ of any alignment is the number of bits of information from New which it encodes, minus the information cost (in bits) of the sequences from Old which are used in the encoding. The latter can be less than the sum of the bits for each of the Old sequence in the alignment because some sequences in Old can encode sequences of two or more other sequences in Old and this recursively through arbitrarily many levels in the manner of hierarchical grammars. When some sequences in Old are encoded in terms of other sequences as just described, the number of bits required overall for the encoding can be reduced.

3.2. Search

Given that the abstract space of possible alignments is normally extremely large, exhaustive search is normally impossible. It is generally recognised that any practical search for good alignments means constraining the search so that parts of the search space (often very large parts) are not entered at all. In broad terms, this may be done in two ways (which need not be mutually exclusive):

- The space of possible alignments may be searched in stages, selecting only the most promising path or paths at each stage and using some measure of 'goodness' (e.g., compression or probability) to guide the selections made at each stage. Versions of this kind of 'metrics-guided' search include 'hill climbing', 'beam search', 'genetic algorithm', 'simulated annealing' and others. With this kind of constraint, large parts of the search space may be pruned away but no part of the search space is ruled out *a priori*. Potentially, any kind of alignment can be reached.

- Arbitrary constraints may be applied so that parts of the search space can never be entered. For example, some arbitrary ceiling may be set on the maximum number of unmatched symbols between hits. This means that it is not necessary to examine any of the alignments containing one or more gaps between hits which exceed the maximum which has been set. With this kind of constraint, a lot of searching can be saved because parts of the search space are ruled out from the beginning. The penalty, of course, is that solutions in those areas - some of which may be good solutions - can never be found.

With either kind of constraint, it is not possible to guarantee that the best possible alignment has been found unless the sequences to be aligned are very few and small. There is a trade-off between the amount of searching required and the 'quality' of the results obtained. It is always possible to make the search manageable by discarding larger parts of the search space. That said, it seems that acceptably good results can often be obtained with relatively modest amounts of searching: a small sacrifice in quality can often mean dramatic savings in the amount of searching required.

There is now a number of methods of searching which give quite reasonably good results with biochemical sequences. Some of these methods are reviewed in Taylor (1988), Barton (1990), Chan, Wong *et al.* (1992) and Day and McMorris (1992). However, it seems that none of the existing methods are adapted to the generalised version of the multiple alignment problem described in Section 3.3 and none of the methods are suitable for finding the kinds of alignments described later in this article. A new search method is described in Wolff (submitted, b).

3.3. A generalised version of the multiple alignment problem

By contrast with most work in bio-informatics, 'multiple alignment' in the present context covers cases where a given sequence may appear two or more times in different parts of an alignment. This includes cases where any one sequence may be aligned with itself (with the obvious restriction that any one symbol should not be aligned with itself). These points are illustrated in some of the examples below.

4. SOME BASIC CONCEPTS IN LOGIC

This section considers how some of the simpler LS concepts may be understood as ICMAUS and, in Section 4.6, describes some possible LR versions of these ideas.

4.1. Logical operators

Each of the basic logical operators, NOT ($\neg$), IMPLIES ($\supset$), AND ($\wedge$), inclusive OR ($\vee$), exclusive

OR, and EQUIVALENT (↔) is defined in LS by a truth table as illustrated by the three examples in Fig. 2.

Please insert Fig. 2 about here

Fig. 2. Examples of truth table definitions of logical operators.

For any given table, a logical operation is performed by searching for an exact match between one or more 'input' values and the values in the 'input' cell or cells of one of the rows in the table. If such a match is found, then the 'output' value may be read from the output cell of the same row. For example, if the input values '0' and '1' (in that order) are applied to the table for IMPLIES, the third row of the table is selected and the output is '1'.

This process of 'table lookup' may be seen as partial pattern matching with unification as described in the Introduction. For example, the IMPLIES operator, may be represented as a set of patterns like this:

```
1 1 ; 1 ;
1 0 ; 0 ;
0 1 ; 1 ;
0 0 ; 1 ;
```

Given this definition and an 'input' pattern like this: '`0 1 ; ;`', the best alignment of the input pattern with one of the patterns in the definition appears to be what is shown in Fig. 3. The corresponding unification is also shown.

Please insert Fig. 3 about here

Fig. 3. Alignment and unification of an 'input' pattern with part of the LS definition of the IMPLIES operation. Symbols which are the result of unification are shown in bold type.

In this and later examples, symbols which are the result of the unification of two (or more) matching symbols are shown in bold type. In the unification shown in this figure, the 'output' is the symbol in plain type which originated in the 'output' part of the definition and has not been unified with any symbol from the 'input'.

In accordance with the remarks in Section 2.2, the semi-colon symbol (';') which appears in the example is *not* a meta symbol with special meaning: it enters into matching and unification like any other symbol. That said, symbols like this have a role to play which is comparable with the role of brackets and other punctuation marks in conventional systems. In an ICMAUS framework, it is necessary to include instances of symbols like ';' in the sequences to be matched to reduce ambiguity and ensure that the 'correct' alignment will give more compression than any of the many incorrect alternatives.

4.2. Simple functions and simple relations

In its simplest form, a 'function' is a table which relates input values to corresponding output values. In its simplest form, a 'relation' is the same except that output values are restricted to the set $\{1, 0\}$. Any of the logical operators considered in the last subsection may be regarded as simple relations.

Any simple function, simple relation or logical operator which is represented in tabular form may be evaluated by table lookup. It should be clear from the example presented in the last subsection that any such case may be interpreted as alignment and unification of patterns.

4.3. Simple rewrite rules

In their simplest form, rewrite rules like this:

green light → go

are very much like the rows in a table representing a function, relation or logical operation. The symbol (or symbols) to the left of the arrow represents potential input and the symbol (or symbols) to the right of the arrow represents potential output.

As before, actual output is obtained by reading the symbol(s) to the right of the arrow when there is a match between actual input symbol(s) and symbol(s) to the left of the arrow. As before, this effect may be seen in terms of the alignment and unification of patterns.

4.3.1. Rewriting

Rules like these are described as 'rewrite' rules because each rule is commonly interpreted as an instruction to delete a pattern of one or more symbols in the input which matches the left side and replace it with a copy of the symbol(s) on the right.

At first sight, this appears to be rather different from alignment and unification of patterns as described above. But if a pattern in the input is deleted and the matching pattern on the left side of a rule is retained, this is equivalent to the merging or unification of the two patterns. And it seems that explicit copying of the symbol(s) on the right side of any rule is not necessary: an equivalent effect may be achieved by reading these 'output' symbols *as if* they had been copied into different contexts.

A relatively full account of how the workings of simple rewrite rules may be understood as ICMAUS is given in Wolff (submitted, b). Examples also appear later in this article (see Sections 4.5.1, 4.6.2 and 5.1).

4.3.2. Reduction

Rewrite rules are sometimes also called 'reduction' rules (Klop, 1992), apparently because, in some applications, rules are applied that have more symbols on the left than on the right. If a structure is processed by these rules so that patterns which match the left sides are replaced by the copies of the corresponding right sides, the size of the structure will be reduced.

This alternative name for rewrite rules seems to be somewhat misleading because there are other applications, especially in theoretical linguistics, where rewrite rules commonly have more symbols on the right than on the left. In cases like these, the application of rewrite rules can have the effect of converting a relatively small structure into a larger one.

Notwithstanding the use of the alternative name 'reduction', it seems that the possible connections between these kinds of rules and concepts of information theory, redundancy and IC as presented in this article (Section 3.1 and elsewhere) are not generally recognised.

4.4. Variables

A variable may be regarded as a receptacle for information which may be empty. Normally, a variable has a name but in some systems, e.g., Prolog, it is possible to specify variables without names which function merely as place markers for missing information.

It seems that the effect of a variable may be modelled with ICMAUS by providing a pair of 'marker' symbols which may be aligned with matching symbols at the start and end of information representing a possible value for the variable. For example, an empty variable, '*x*', may be represented by the pair of contiguous symbols, 'x ;', within a larger sequence; possible 'values' for this 'variable' may be flanked by matching symbols as, for example, in 'x 1 ;' or 'x 0 ;'. Fig. 4 shows how such a 'variable' may acquire a 'value' by alignment and unification of patterns.

Please insert Fig. 4 about here

Fig. 4. Alignment and unification of a 'variable' with a 'value'. The ellipses ('...') represent other symbols which may lie to the left and right of 'x ;'.

The similarity between Fig. 4 and Fig. 3 will not have escaped the reader. In Fig. 3, the pair of symbols '; ;' in '0 1 ; ;' may be seen to function as an unnamed variable which receives '1' as its 'value'.

4.5. Types for variables

In the example just considered, it seems that there are two alternative 'best' alignments (with corresponding unifications): the one shown in Fig. 4 and one where 'x ;' is aligned and unified with 'x 0 ;'. Thus the patterns 'x 1 ;' and 'x 0 ;' may be seen as providing a definition of the set of possible 'values' for the 'variable' 'x ;'. In other words, they provide a 'type definition' for the 'variable' which, in this case, is a definition of the Boolean type.

4.5.1. Types with Infinite Range

In many cases, of course, the type of a variable covers an infinite range of values, e.g., 'integer', 'real', 'string' and others. Often, a type of this kind may be defined using simple rewrite rules. For example, strings of alphabet characters may be defined with rules like these:

§ →
§ → a §
...
§ → z §
§ → A §
...
§ → Z §

The first rule ('§ →') expresses the idea that a string may be empty. The subsequent rules define recursively the infinite set of alphabetic strings containing one or more letters.

In keeping with the ideas described above, the rewrite rules just shown may be translated into the following patterns:

§ ;
§ a §
...
§ z §
§ A §
...
§ Z §

Given these Old patterns, together with an Old pattern, '1 a b c § ; d e f #' which contains a 'variable', and a New 'input' string ('a b c x y z d e f'), the best alignment appears to be what is shown in Fig. 5.

Please insert Fig. 5 about here

Fig. 5. An alignment amongst patterns comprising an alphabetic string ('a b c x y z d e f'), a pattern containing a 'variable' ('1 a b c § ; d e f #') and patterns which constitute a 'type definition' of the variable.

The corresponding unification is:

1 a b c § x § y § z § ; d e f #

Notwithstanding the inclusion of instances of the 'service' symbols,'1', '§', ';' and '#', this unification achieves the effect of an 'assignment' of the alphabetic string to the 'variable'.

Compression of the 'New' input string 'a b c x y z d e f' is achieved because it is, in effect, recognised as an instance of a pre-established 'Old' string, '1 a b c § ; d e f #', with the interpolation of 'x y z' in the position of the variable. Thus the New string may be encoded as '1 x y z #'. Assuming that the identifier '1' is encoded with fewer bits than 'a b c' and that the termination symbol, '#', requires fewer bits than 'x y z', the encoding of the New string clearly requires fewer bits than the original.

Why is it necessary to include the symbols '§ ;' in the middle of '1 a b c § ; d e f #', and to include patterns like '§ a §', '§ b §' and so on in the set of Old patterns? Without these symbols and patterns, it would not be possible to encode '1 a b c § ; d e f #' as '1 x y z #' because there would be ambiguity about the relative positions of the symbols 'a', 'b', 'c', 'x', 'y', 'z', 'd', 'e' and 'f'. The 'service' symbols and patterns serve as 'anchors' which ensure that the relative positions of the alphabetic symbols are preserved.

4.5.2. Separating the Declarations of Types and Variables

The two examples of 'type definitions' given so far differ from type definitions in conventional systems because, in each case, the definition contains the name of the variable whose type is to be specified. The inflexibility of an arrangement like this is avoided, in conventional systems, by defining each type independently of any specific variable and then connecting it to one or more variables in one or more declarations like '*int i*' or '*float r*'.

To see how this idea may be modelled with ICMAUS, consider a function whose name and formal parameter is declared as '*g*(*boolean x*)'. This declaration, and the definition of the Boolean type, may be represented with the following patterns:

```
g ( boolean x # ; )

boolean # 1 ;
boolean # 0 ;
```

In this case, the definition of the Boolean type provided by the second and third patterns does not contain the name, 'x', of the variable. The symbol 'boolean' provides the link between the variable declaration and the type definition. Any other variable may be linked to the type definition in the same way.

If *g*(*x*) is 'called' from some context like this: '... *g*(1) ...', the actual parameter ('1') is assigned to the formal parameter ('*x*'). The way in which this may be modelled by alignment and unification of patterns is shown in Fig. 6.

Please insert Fig. 6 about here

Fig. 6. Alignment and unification of patterns for the assignment of an 'actual parameter' to a 'formal parameter'.

In this example, '#' may be regarded as a 'service' symbol whose function is to ensure that a 'correct' alignment is achieved. It seems that its interpolation between the left and right markers of the 'variable', 'x ;', does not invalidate the assignment and would not prevent the value of the variable being 'transferred' to other contexts as described in Section 5, below.

4.5.3. Universal and Existential Quantification

In many cases, a formula in first order logic which contains one or more instances of the universal quantifier ('∀') or the existential quantifier ('∃') or both may be converted into an equivalent 'clausal' form in which the quantifier(s) have been omitted (Eisinger and Ohlbach, 1993). Systems like Prolog appear to provide much of the 'power' of first order logic without the need for explicit quantification of variables.

Accordingly, no attempt has been made in this programme of research to model the universal and existential quantifiers explicitly. It is assumed as a working hypothesis that concepts of quantification may be accommodated in the ICMAUS framework in a manner comparable with their treatment in Prolog. That said, it should be emphasised again that the ICMAUS framework is *not* merely a syntactic variation on Prolog, that it lends itself to tasks requiring fuzzy matching in ways which would be cumbersome to achieve with Prolog.

4.6. Possible rationalisations of basic concepts

This section considers how, within the proposed framework, there may be some scope for replacing some of the basic LS concepts with simpler alternatives. As previously noted, while the ICMAUS framework provides a home for these LR concepts, some of them are similar to concepts which are already recognised in, for example, Prolog. The following subsections discuss some of these possibilities.

4.6.1. AND

Within the concept of a string, sequence or 'pattern' of symbols (which is part of the foundation of the ICMAUS proposals) the relationship between a symbol and any of its neighbours may be regarded as an example of the concept 'AND'. In accordance with the truth-table definition of the concept, the AND relationship between any two symbols applies only if they are both present in a given pattern.

If it is accepted that AND is an implicit primitive within the concept of a sequence of symbols, it seems unnecessarily cumbersome to interpret the concept always in terms of its truth table or an equivalent set of patterns. Any conjunction of propositions like '$p \wedge q$' may be converted into something like '$p\ q$', where the space between the two propositions represents the primitive concept of AND, replacing the LS concept represented by '$\wedge$'.

An example of how this more primitive concept of conjunction may be applied in the ICMAUS framework is presented in Section 6.2.

4.6.2. Exclusive OR

Some concept of disjunction is embodied in the idea that, within the ICMAUS framework, there may be two or more *alternative* alignments (with corresponding unifications) for any given set of patterns and, more specifically, that amongst these alternative alignments, there may be two or more alternative 'best' alignments.

This idea appeared in Section 4.5, above, where it was suggested that the two alternative values of a variable with a Boolean type might be modelled by two alternative 'best' alignments and unifications of patterns.

In a similar way, it seems that disjunction in the operation of rewrite rules, may be modelled by alternative alignments of patterns. Consider, for example, some 'linguistic' rules like these:

NP → D N
D → the ∨ a ∨ some ∨ one ∨ ...
...

In the context of the first rule, the second rule may be read as a statement that the ‘determiner’ at the beginning of a ‘noun phrase’ may be realised as ‘the’ or ‘a’ or ‘some’ or ‘one’ etc.

These rules may also be written like this:

NP → D N
D → the
D → a
D → some
D → one
...

And then translated into patterns like this:

NP D ; N ; ;
D the ;
D a ;
D some ;
D one ;
...

It seems that the alternatives in the rewrite rules may be modelled by alternative ‘best’ alignments and unifications between the pattern ‘D ;’ within the pattern ‘NP D ; N ; ;’and each of the ‘D ... ;’ patterns representing specific words. In keeping with the discussion in Sections 4.4 and 4.5, the former may be regarded as a ‘variable’ and the latter may be seen as a ‘type definition’ of the ‘variable’.

Modelling disjunction in this way is similar to the way in which two or more clauses within a Prolog procedure represent alternative realisations of that procedure.

These examples seem to correspond to the ‘exclusive’ version of the OR relation because one of the alternatives is normally chosen, and only one. The way in which an LR version of the inclusive OR concept might be modelled in terms of ICMAUS is not, at present, clear.

4.6.3. IMPLIES

It is generally understood that the LS concept of ‘material implication’ ($p \supset q$), otherwise defined as $\neg(p \wedge \neg q)$, is not the only meaning of ‘implication’ (Copi, 1986). And, of the several possible meanings, it appears to be one of the least ‘natural’. Certainly, it can lead to such counter-intuitive statements as: “If a statement is false then it materially implies any statement whatever” (Copi, 1986).

In the Introduction, it was suggested that human capabilities for recognising patterns when they are partially obscured or fragmented supports a kind of ‘naturalistic’ inference or implication: recognition of a pattern from a subset of its parts means that the unseen parts are, in effect, inferred or implied.

One of the strengths of the ICMAUS framework is that it can model the recognition of patterns from subsets of their parts. Thus it may also provide a means of modelling a form of inference or implication which appears to be more natural than material implication. An example is presented in Section 6, below.

This alternative view of implication as partial pattern matching suggests that the LR version of the IMPLIES relation is essentially the same as the LR version of the AND relation: if two symbols or patterns of symbols are found together within a larger pattern then there is mutual implication and a mutual AND relationship between them.

4.6.4. TRUE and FALSE

In standard treatments of logic, propositions normally have explicit values for TRUE and FALSE. This kind of explicit marking of positive and negative values may also be used in computing applications like commercial databases. For example, a database for railway bookings, may record whether a seat faces forwards or backwards with “Seat faces forward: yes/no” or, equivalently, “Seat faces: F/B”.

However, some compression can normally be achieved if a default value is assumed in every case which is not marked otherwise. Thus seats may be assumed to face forwards unless marked “B” or they may be assumed to be second class unless marked “First”. When this kind of technique is used, most compression is achieved if the default value is the one which occurs most frequently, i.e., is most probable. In the case of seats for smokers or non-smokers, the latter is the most frequent in UK railway carriages today and should be chosen as the default value. Ten or twenty years ago, the opposite was true. When Nixon said “I am not a crook”, he was acknowledging the prior expectation that he was.

In many applications the imbalance between TRUE and FALSE cases is so extreme and it would be very inefficient to use explicit values for TRUE and FALSE. For example, any travel agent’s database of flights between cities would normally store only those which are available, not all the multitude of possible flights which are *not* available. If the database is queried for a particular flight and no match is found, this is normally interpreted to mean that the flight is not available (although it could mean that some mistake has been made in entering the available flights).

The practice of assigning the value TRUE to any query which finds a match and FALSE to any query which fails to find a match is, of course, the principle of ‘negation as failure’, which has a key role in the operation of Prolog. In the light of the foregoing remarks, negation as failure may be seen as an extreme example of how compression may be achieved by the use of default values.

In general, there seems to be a closer connection than is normally recognised between IC and concepts associated with TRUE and FALSE.

ICMAUS may be used in a ‘negation as failure’ mode when that is appropriate. And, like Prolog, ICMAUS will accommodate explicit marking of TRUE and FALSE if that is required (see Section 5, below). As discussed in Section 7.1, the framework seems also to allow the provision of default values which can be overridden with alternative values if that is required.

5. COMPOSITION OF LOGICAL STRUCTURES

Logical formulae may be composed, recursively, from simpler formulae as shown by these rewrite rules:

f → 'atomic formula'
$f \rightarrow (\neg f)$
$f \rightarrow (f \supset f)$
$f \rightarrow (f \wedge f)$
$f \rightarrow (f \vee f)$
$f \rightarrow (f \leftrightarrow f)$

In a similar way, an atomic formula is composed of a relation symbol ('*r*') with one or more 'terms', arranged like this:

$r(t_1, t_2, ..., t_n)$.

And, amongst the terms, there may be 'functions' which may be composed recursively of simpler structures.

5.1. Composition: syntax

At the syntactic level, recursive composition of structures may be modelled in an ICMAUS framework in much the same manner as can be done with rewrite rules. The rules, above, may be translated as:

```
( f af )
( f not ( f ) )
( f ( f ) implies ( f ) )
( f ( f ) and ( f ) )
( f ( f ) or ( f ) )
( f ( f ) equiv ( f ) )
```

and each instance of the trio of contiguous symbols, '(f)', may be expanded by alignment and unification with any of the patterns which start with '(f' and ends with ')' (cf. Section 4.6.2, above. See also Wolff (submitted, b)). Fig. 7 shows how a composite formula

```
((not (af implies af)) and af)
```

(in which 'af' stands for 'atomic formula') may be aligned with the above patterns.

Please insert Fig. 7 about here

Fig. 7. An alignment between a composite formula and some of the patterns defining the structure of formulae.

5.2. Composition: semantics

When the syntactic forms in a composite structure have explicit semantic values, there is a need to ensure that values can be 'passed' from one structure to another so that the overall value of a composite structure may be found. To see how this may be achieved with ICMAUS, consider first how it may be done with Prolog.

For the sake of clarity and because alignments can otherwise become very large, a very simple LS example will be considered, the negation of an atomic formula: '¬canfly(x)'. Although the example is simple, it captures all the essentials of how values may be passed from one structure to another and it should be clear from this example how the ideas will generalise to more complex cases.

The LS version of '¬canfly(x)' may be expressed in Prolog with a set of clauses like these:

```
not_canfly(X, V) :- canfly(X, T), not(T, V).

canfly(robin, 1).
canfly(penguin, 0).

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

If a query is applied like:

```
?- not_canfly(penguin, V).
```

the variable 'X' in the first line is instantiated as 'penguin', the variable 'T' is instantiated as '0' and the variable 'V' is instantiated as '1' - which is the correct result.

For present purposes, the key point is that, in accordance with the fundamentals of Prolog, the two appearances of 'T' in the first line represent a single variable with a single value so that its instantiation as '0' has the effect of 'passing' that value from 'canfly(X, T)', to 'not(T, V)'.

To achieve this kind of effect with ICMAUS, the Prolog clauses may be translated into patterns like these:

```
not_canfly X ; V ; canfly X ; T ; % not T ; V ; % $

canfly robin 1 %
canfly penguin 0 %

not 1 0 %
not 0 1 %

X robin ;
X penguin ;
T 1 ;
T 0 ;
V 1 ;
V 0 ;
```

In this translation, the last six patterns may be regarded as type definitions of the 'variables', 'X ;', 'T ;' and 'V ;' (as described in Section 4.4).

The query '?- not_canfly(penguin, V).' may be translated as:

```
not_canfly penguin V ; $
```

To simplify the alignment for the sake of clarity, the first pattern in the translated 'program' above, will be reduced to 'not_canfly canfly X ; T ; % not T ; V ; % $'.

Fig. 8 shows what appears to be the best alignment between the 'query' pattern and the (reduced) patterns defining the negation of the 'canfly' relation. Notice that the pattern 'not_canfly canfly X ; T ; % not T ; V ; % $' appears twice in the alignment, in accordance with the generalisation of the alignment problem described in Section 3.3. Notice also that although each symbol in this pattern appears twice in the figure - which suggests the possibility of matching and alignment - no such symbol can be matched and aligned with itself.

Please insert Fig. 8 about here

Fig. 8. An alignment of patterns showing how a 'value' may be 'passed' between two instances of a 'variable'.

In Fig. 8, the symbol 'penguin' in the 'query' has the effect of selecting the pattern 'canfly penguin 0 %' so that, of the two main contenders for alignment with 'T ;', 'T 0 ;' is selected and, in effect, 'T ;' receives '0' as its 'value'. Since the two instances of 'T ;' are aligned with each other, that 'value' is, in effect, 'passed' from 'canfly X ; T ; %' to 'not T ; V ; %'. Then that value of 'T ;' selects 'not 0 1 %' so that, in effect, 'V ;' receives '1' as its value. This, of course, is the 'correct' result.

The foregoing analysis may seem to create undue complexity out of the apparent simplicity of '$\neg p$'. But, as noted in Section 2.2, any such impression is probably a matter more of appearance than reality. Any other attempt to analyse the composition of LS structures, including their semantics, in terms of primitive operations is likely to yield similar complexity.

It seems that the analysis should generalise to any combination of relations, functions and logical operators and the passing of values amongst such structures.

6. DEDUCTION

This section considers both LS and putative LR forms of deduction. The latter provides a natural introduction to the area of nonmonotonic and uncertain reasoning considered in the following section.

6.1. Modelling 'standard' forms of deduction

$\forall x$: bird(x) $\supset$ canfly(x)
bird(Tweety)
$\therefore$ canfly(Tweety).

Rather inelegantly, LS versions of the two premises of this *modus ponens* form of syllogistic reasoning may be modelled in Prolog like this:

```
bird_implication(X, V1, V2) :-
        bird(X, V1),
        canfly(X, V2),
        implies(V1, V2, 1).

bird(tweety, 1).

canfly(_, 1).

implies(1, 1, 1).
implies(1, 0, 0).
implies(0, 1, 1).
implies(0, 0, 1).
```

Given the query '?- bird_implication(X, V1, V2).' (which would be true), the variable 'X' is instantiated as 'tweety', the variable 'V2' is instantiated as '1', thus confirming 'canfly(tweety, 1)', i.e., that Tweety can fly.

Given the discussion in Section 5.2, above, it appears that this treatment of *modus ponens* reasoning, with explicit values for TRUE and FALSE, could, in principle, be modelled as ICMAUS. No attempt will be made here to show the corresponding alignments because they would take too much space.

Since any of the other basic forms of LS reasoning (*modus tollens*, constructive dilemma, disjunctive syllogism etc.) may be modelled in Prolog in a similar way, it appears that, in principle, these may also be modelled as ICMAUS.

6.2. Alternatives to standard forms of deduction

Of course, Prolog, as it is normally used, takes advantage of 'negation as failure' to achieve a significant simplification of *modus ponens* and other forms of reasoning. For example, the implication that if something is a bird then that something can fly would normally be expressed in Prolog, in reversed form, like this:

canfly(X) :- bird(X).

This version of the implication illustrates some of the ideas described in the Introduction and in Section 4.6. The clause 'canfly(X) :- bird(X).' may be seen simply as a 'pattern' linking the ability to fly with the concept of a bird. In this pattern, the symbol ':-' is largely redundant. The implication may be seen as partial pattern matching: if 'bird(X)' is successfully matched then we may infer 'canfly(X)'. If the implication is viewed in this way, then, as suggested in Section 4.6.3, the distinction between IMPLIES and AND, in their 'naturalistic' interpretations, becomes blurred.

These observations might suggest that the way forward in any attempt to develop LR versions of logic would be to translate Prolog clauses like 'canfly(X) :- bird(X).' into patterns like this:

canfly X ; bird X ;

and to model the workings of Prolog using ICMAUS. The discussion in Section 5.2 suggests that this is possible and that, for example, ICMAUS could model the 'passing' of values between the two instances of 'X' in clauses like 'canfly(X) :- bird(X).'.

But the flexibility of the ICMAUS framework suggest that there may be better opportunities than this. Knowledge that (most) birds can fly is simply one item of information amongst the many things that we know about birds: that they have wings, feathers, beak, crop, that they lay eggs, that some birds have an individual name, and so on. It seems unnecessarily cumbersome to record each of these pieces of information as a separate clause like 'canfly(X) :- bird(X).'. It seems much more natural to record them all together in a single pattern like this:

bird name ; canfly ; wings ; feathers ; beak ; crop ; lays_eggs ; ... $

In accordance with the remarks in Section 2.2 about the relationship between sensory data and stored knowledge, this kind of representation seems to reflect empirical input much more directly than a set of separate Prolog clauses. With a representation of knowledge like this, there seems to be a better chance of integrating logic with other aspects of cognition such as pattern recognition or learning.

With this kind of description of a typical bird, we may suppose that, for each 'field' in the description (an alphabetic symbol followed by ';'), there are zero or more other patterns which expand the information for that field, in the same way that symbols in a rewrite rule may be expanded by other rewrite rules. For example, typical (pet) names for birds may be specified such as 'name Tweety ;', 'name Chirpy ;' and so on.

Given patterns like these, the existence of a particular bird, Tweety, may be specified with a pattern like this:

bird Tweety $

And the inference that Tweety can fly may be made from what appears to be the best alignment and unification amongst these patterns, shown in Fig. 9.

Please insert Fig. 9 about here

Fig. 9. Alignment and unification of patterns related to birds. The ellipsis ('...') represents other characteristics of birds which cannot be listed in the space available.

There are (at least) three points of interest here:

- With this version of the relationship between being a bird and the ability to fly, there is no need to repeat the 'variable', 'name ;', as is done with the corresponding variable, 'X', in the Prolog version. The possibility that a bird has a name is just one of the many features of any bird which may be stored once in a pattern alongside those other features.

- The probability that Tweety can fly is not the only inference that may be made from the alignment and unification shown in Fig. 9. We may also conclude that Tweety has wings, feathers and all the other characteristics of birds. Given the information that Tweety is a bird, it seems much more natural to make all those inferences in one operation than to make each inference separately as seems to be required with some LS approaches to this kind of problem. Notice that this observation is independent of whether or not the operation is done in series or in parallel. This is because completion of the unification (by serial or parallel processes) leads at once to the inference that Tweety has all the attributes listed in the pattern describing the concept of a bird.

- The given inference is an example of *inheritance* of the attributes of a class by an instance of the class. (In other cases, of course, the attributes of a class may be inherited by a sub-class of the given class). This form of reasoning has been incorporated in several programming languages (originally Simula (Birtwistle *et al*., 1973), followed by Smalltalk, C++ and others) and has been the subject of other research more closely related to logic (see, for example, (Horty, 1994)).

This last point will be picked up again in Section 7 where ICMAUS proposals like the one just outlined are developed in the context of nonmonotonic and uncertain reasoning.

6.3. Chains of reasoning

Before leaving this section, it is appropriate to mention the idea which is discussed more fully elsewhere (Wolff, 1996, submitted, a), that ICMAUS may accommodate the 'chaining' of reasoning which is a familiar feature of everyday reasoning and, indeed, of LS reasoning and Prolog.

If A implies B and B implies C then we may conclude that A implies C - and likewise when the chains are longer. In keeping with the discussion in Section 6.2, above, each of the implications in any chain like this may be represented as a simple association of symbols in a pattern: 'A B', 'B

C', 'smoke fire', 'fire burn', and so on. It is not hard to see that such patterns may be linked together by ICMAUS and chains of reasoning may thereby be modelled.

By contrast with the relatively rigid chaining embodied in Prolog, the ICMAUS framework can accommodate more flexible linking of patterns which is, perhaps, more in keeping with human thinking. Relevant examples are presented in (Wolff, 1996, submitted, a).

7. NONMONOTONIC AND UNCERTAIN REASONING

Although concepts associated with propositional logic and first-order logic provide a tightly organised and internally consistent view of logic, they seem to lack the flexibility needed to capture the relaxed style of everyday reasoning where uncertainty is the rule and where something which is true in one context may easily be overridden in another. This section describes in outline how the ICMAUS concepts may be applied to three ideas which seem to be closer in spirit to this more flexible kind of reasoning: inheritance of attributes, abductive reasoning and inductive reasoning. Each of these are large subjects so, in the space available, only a sketch of the possibilities can be given.

7.1. Inheritance and default reasoning

This section considers how the ICMAUS framework may accommodate three aspects of inheritance: the inheritance of attributes through two or more levels, cross classification or 'multiple inheritance', and the way in which default attributes may be negated or overridden in specific contexts.

7.1.1. Inheritance Through Two or More Levels

In the example given in Section 6.2, Tweety inherits attributes directly from the single class 'birds'. But it is widely recognised that classes may form hierarchies which may be arbitrarily deep and that a given entity may inherit attributes from all levels in such a hierarchy. In Fig. 10, the pattern at the top which describes Tweety as a swallow is aligned with other patterns so that, as before, Tweety may be seen to inherit the general characteristics of birds and, in addition, Tweety inherits characteristics which are specific to swallows: that they can fly long distances and that they have pointed wings.

Please insert Fig. 10 about here

Fig. 10. Alignment and unification of patterns modelling the inheritance of attributes in a two-level hierarchy.

It should be clear from this example how ICMAUS could accommodate the inheritance of attributes through class hierarchies which are arbitrarily deep.

7.1.2. Cross-Classification or 'Multiple Inheritance'

A given entity may belong in two or more classes which are not sub-classes, one of another.

Amongst living things, the classes ‘male’ and ‘female’ cut across classes such as ‘bird’, ‘fish’, ‘mammal’ and so on. The way in which ICMAUS may accommodate this kind of cross-classification or ‘multiple inheritance’ is suggested by the example in Fig. 11 where a living thing called ‘Chirpy’ (let’s give Tweety a rest!) inherits attributes from a pattern representing the class ‘male’ and, at the same time, inherits attributes from the class ‘bird’.

Please insert Fig. 11 about here

Fig. 11. Alignment and unification of patterns modelling multiple inheritance.

This example suggests that gender characteristics are totally independent of classifications like ‘bird’, ‘fish’ and so on, but of course we know that they combine in complex and subtle ways. The example in Fig. 10 suggests how attributes may combine in an ICMAUS framework but more work is needed in this area.

7.1.3. Negation of a Default Attribute

The chief role of the overworked ‘Tweety’ in discussions like these is to be a bird which, by default, has the ability to fly but loses this ability if we learn that Tweety is a penguin. Although this kind of adjustment of our expectations is a familiar part of everyday reasoning, it is a problem for LS which has the property of *monotonicity*: any conclusion which is valid in one state of knowledge remains valid whatever new knowledge is acquired (Ginsberg, 1994).

It seems that ICMAUS may provide a framework in which the nonmonotonicity of everyday reasoning may be accommodated. In Fig. 12, the second pattern in the alignment describes a penguin as a bird in which the attribute ‘canfly’ has been qualified with ‘no’ (and the attribute ‘wings’ has been qualified with ‘stubby’). Given the information that Tweety is a penguin (in the top pattern), it seems that alignment and unification of patterns achieves the effect of negating the expectation that Tweety would have the ability to fly (and also leads us to expect that Tweety’s wings would be ‘stubby’).

Please insert Fig. 12 about here

Fig. 12. Alignment and unification of patterns modelling the negation of an attribute.

Assuming that the pattern describing birds remains in the store of knowledge, then knowing that penguins cannot fly should not disturb the expectations, derived by alignment and unification with the pattern for birds, that other birds - swallows, guillemots, parrots and so on - can fly.

The similarity between Fig. 12 and Fig. 10 suggests that negation of an attribute does not require special status and may be treated in exactly the same way as any other qualifier of an attribute in an inheritance hierarchy.

7.2. Abductive reasoning

If we know that something can fly, it is not unreasonable to think that it might be a bird. It could, of course, be an aeroplane, a bat, a flying squirrel and so on but, given the world as we know it, there is a good chance that it would be a bird.

Given the LS implication that all birds can fly, this is back-to-front 'abductive' reasoning which does not fit easily into an LS framework. The suggestion here is that ICMAUS may offer a better home for this kind of reasoning which, like nonmonotonic reasoning, is a prominent feature of how we ordinarily think.

To avoid an *ad hoc* treatment for inductive reasoning within the ICMAUS framework, it would be natural to express the association between being a bird and being able to fly in a simple pattern like the one shown in Fig. 9 which includes all the other attributes normally associated with birds. The alignment and unification in Fig. 13 suggests how, if we know only that 'Tweety' can fly, we may infer that he, she or it is a bird - and we may also infer that this entity has wings, feathers and other attributes of birds. Of course, with this example, we must discount any other knowledge we may happen to have about the kinds of names that birds may be given.

Please insert Fig. 13 about here

Fig. 13. An abductive inference as alignment and unification of patterns.

The sceptical reader may object that the abductive inference modelled in Fig. 13 says more than we are entitled to extract from the original implication that "All birds can fly". Contrary to what seems reasonable, the right-to-left inference suggests that anything which can fly is a bird!

The suggested answer to this objection is a reminder that all inferences in an ICMAUS framework are probabilistic and the observation that the probability of any inference depends on the knowledge which is stored.

If the system only knows one pattern and that pattern shows an association between being a bird and having the ability to fly then it would indeed infer with a high probability that anything which can fly is a bird. A little reflection suggests that if our knowledge was equally restricted then we would probably reach the same conclusion.

But if a more realistic range of patterns is stored, including the facts that aeroplanes, bats and some kinds of squirrels can fly, then the knowledge that some entity can fly should lead to a set of alternative alignments highlighting the things which are known to be able to fly but without ascribing a high probability to any one of these inferences.

In summary, the suggestion is that the asymmetry in the normal LS formulation of implication may, with some profit, be replaced by a system of inference in which knowledge is stored simply as 'patterns' and that asymmetries associated with inferences, if they exist, emerge from the probabilities associated with each alignment which, in turn, depend on the constellation of patterns which is stored at any one time.

The potential profit from adopting this standpoint is a uniform treatment of deduction and abduction within a framework which apparently has potential to assimilate other aspects of logic and computing.

The foregoing is, of course, only a sketch of how abductive reasoning may be modelled with ICMAUS. Future work will seek to give quantitative precision to the ideas outlined here.

7.3. Inductive reasoning

In Wolff (1996), it is suggested that there is a closer connection between inductive reasoning and deductive reasoning than has traditionally been thought. Information may be compressed by pattern matching, unification and search and, as a by-product, recurrent patterns may be identified (Wolff, 1995b). Then by partial pattern recognition (which is also IC by pattern matching, unification and search), the observed parts of a recurrent pattern may lead to the prediction or inference of the unseen parts.

It should be clear from the foregoing discussion that both these processes of pattern matching, unification and search may be modelled as ICMAUS. The identification of recurrent patterns may be seen as unsupervised inductive learning. Examples are described in Wolff (1991, 1995b). The partial matching and merging of patterns, with a corresponding inference of the unmatched parts, is illustrated by the examples described in Sections 4.1 and 6.2, above (see also Wolff, 1996).

8. CONCLUSION

This article has described in outline how IC by *multiple alignment* (with unification and search) may accommodate varied aspects of logic in its ‘standard’ form and at the same time offers the potential for simplification and rationalisation in some areas of logic. The proposed framework seems to be particularly well suited to the relatively flexible kind of reasoning which we use in everyday thinking which is characteristically nonmonotonic and which rarely has the all-or-nothing certainty of standard logic.

These proposals originate in a programme of research which seeks to integrate a wide range of concepts in computing and cognition within a framework of information compression by multiple alignment, unification and search.

There are many questions still to be answered about these proposals. I hope that what has been presented has shown sufficient promise to encourage other researchers to develop these ideas and explore their potential applications.

ACKNOWLEDGEMENTS

I am grateful to John Hornsby of the School of Electronic Engineering and Computing Systems, Bangor, for advice on mathematical formulae and to anonymous referees for comments on a previous version of the paper.

REFERENCES

Abramsky, S., Gabbay, D. M. and Maibaum, T. S. E. (eds.) (1992) *Handbook of Logic in Computer Science*, Vols. 1-4. Clarendon Press, Oxford.

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

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

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 Biology*, 39, 418-430.

Attneave F (1954) Informational aspects of visual perception. *Psychological Review*, 61, 183-193.

Barlow, H. B. (1969) Trigger features, adaptation and economy of impulses. In K. N. Leibovic (ed), *Information Processing in the Nervous System*. Springer, New York, pp. 209-230.

Barton, G. J. (1990) Protein multiple sequence alignment and flexible pattern matching. *Methods in Enzymology*, 183, 403-428.

Birtwistle, G. M., Dahl, O-J., Myhrhaug, B. and Nygaard, K. (1973) *Simula Begin*. Van Nostrand Reinhold, New York.

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

Cook, C. M. and Rosenfeld, A. (1976) Some experiments in grammatical inference. In J. C. Simon (ed.), *Computer Oriented Learning Processes*. Noordhoff, Leyden, pp 157-174.

Copi, I. M. (1986*) Introduction to Logic*. Seventh Edition. MacMillan, New York.

Davis, M. (1993) First order logic. In Gabbay, Hogger and Robinson (eds.) (1993-4), Vol. 1, 31-65.

Day, W. H. E. and McMorris, F. R. (1992) Critical comparison of consensus methods for molecular sequences. *Nucleic Acids Research*, 20 (5), 1093-1099.

Eisinger, N. and Ohlbach, H. J. (1993) Deduction systems based on resolution. In Gabbay, Hogger and Robinson (eds.) (1993-4), Vol. 1, 183-271.

Gabbay, D. M., Hogger, C. J. and Robinson, J. A. (eds.) (1993-4*) Handbook of Logic in Artificial Intelligence and Logic Programming*. Vols. 1-3. Clarendon Press, Oxford.

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.

Ginsberg, M. L. (1994) AI and nonmonotonic reasoning. In Gabbay, Hogger and Robinson (eds.)

(1993-4), Vol. 3, 1-33.

Horty, J. F. (1994) Some direct theories of nonmonotonic inheritance. In Gabbay, Hogger and Robinson (eds.) (1993-4), Vol. 3, 111-187.

Huffman, D. A. (1952) A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40, 1098-1101.

Klir, G. and Yuan, B. (1995) *Fuzzy sets and fuzzy logic: theory and applications*. Prentice-Hall, Upper Saddle River, NJ.

Klop, J. W. (1992) Term rewriting systems. In Abramsky., Gabbay and Maibaum. (eds.) (1992), Vol. 2, 1-116.

Kosko, B. (1994) *Fuzzy thinking: the new science of fuzzy logic*. Flamingo, London.

Li, M. and Vitányi, P. (1993) An introduction to Kolmogorov complexity and its applications. Springer-Verlag, Heidelberg.

Rissanen, J. (1987) Stochastic complexity. *Journal of the Royal Statistical Society B,* 49(3), 223-239.

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

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

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

Von Békésy G (1967) *Sensory Inhibition*, Princeton University Press, Princeton, NJ.

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

Wallace, C. S. and Freeman, P. R. (1987) Estimation and inference by compact coding. *Journal of the Royal Statistical Socieity B,* 49(3), 240-252.

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

Watanabe, S. (1985) *Pattern Recognition: Human and Mechanical*. Wiley, New York.

Wolff, J. G. (submitted, a) Multiple alignment and computing.

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

Wolff, J. G. (1996) Learning and reasoning as information compression by multiple alignment, unification and search. In A. Gammerman (ed.), *Computational Learning and Probabilistic Reasoning*, Wiley, New York, 67-85. (This is a revised version of the paper 'Learning and reasoning as compression by multiple alignment and unification,' presented at Applied Decision Technologies '95 (ADT '95), Brunel University, London, April 1995, Stream 1 (Computational Learning and Probabilistic Reasoning), 1995, 223-236.)

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

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

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

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

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

Wolff, J. G. (1990) Simplicity and power: some unifying ideas in computing. *Computer Journal,* 33 (6), 518-534. (Reproduced in Wolff (1991), Chapter 4).

Zipf, G. K. (1949) *Human Behaviour and the Principle of Least Effort*. Addison-Wesley, Cambridge, Mass.

FIGURE CAPTIONS

Fig. 1. A scene in which some objects are partially obscured by other objects.

Fig. 2. Examples of truth table definitions of logical operators.

Fig. 3. Alignment and unification of an 'input' pattern with part of the LS definition of the IMPLIES operation. Symbols which are the result of unification are shown in bold type.

Fig. 4. Alignment and unification of a 'variable' with a 'value'. The ellipses ('...') represent other symbols which may lie to the left and right of 'x ;'.

Fig. 5. An alignment amongst patterns comprising an alphabetic string ('a b c x y z d e f'), a pattern containing a variable ('1 a b c § ; d e f #') and patterns which constitute a 'type definition' of the variable.

Fig. 6. Alignment and unification of patterns for the assignment of an 'actual parameter' to a 'formal parameter'.

Fig. 7. An alignment between a composite formula and some of the patterns defining the structure of formulae.

Fig. 8. An alignment of patterns showing how a 'value' may be 'passed' between two instances of a 'variable'.

Fig. 9. Alignment and unification of patterns related to birds. The ellipsis ('...') represents other characteristics of birds which cannot be listed in the space available.

Fig. 10. Alignment and unification of patterns modelling the inheritance of attributes in a two-level hierarchy.

Fig. 11. Alignment and unification of patterns modelling multiple inheritance.

Fig. 12. Alignment and unification of patterns modelling the negation of an attribute.

Fig. 13. An abductive inference as alignment and unification of patterns.

Fig. 1

Fig. 2

AND

Input 1	Input 2	Output
1	1	1
1	0	0
0	1	0
0	0	0

IMPLIES

Input 1	Input 2	Output
1	1	1
1	0	0
0	1	1
0	0	1

NOT

Input	Output
1	0
0	1

Fig. 3

Alignment:

$$\begin{array}{ccccc} 0 & 1 & ; & & ; \\ | & | & | & & | \\ 0 & 1 & ; & 1 & ; \end{array}$$

Unification:

0 1 ; 1 **;**

Fig. 4

Alignment:

```
...  x     ;  ...
     |     |
     x  1  ;
```

Unification:

... **x** 1 **;** ...

Fig. 5

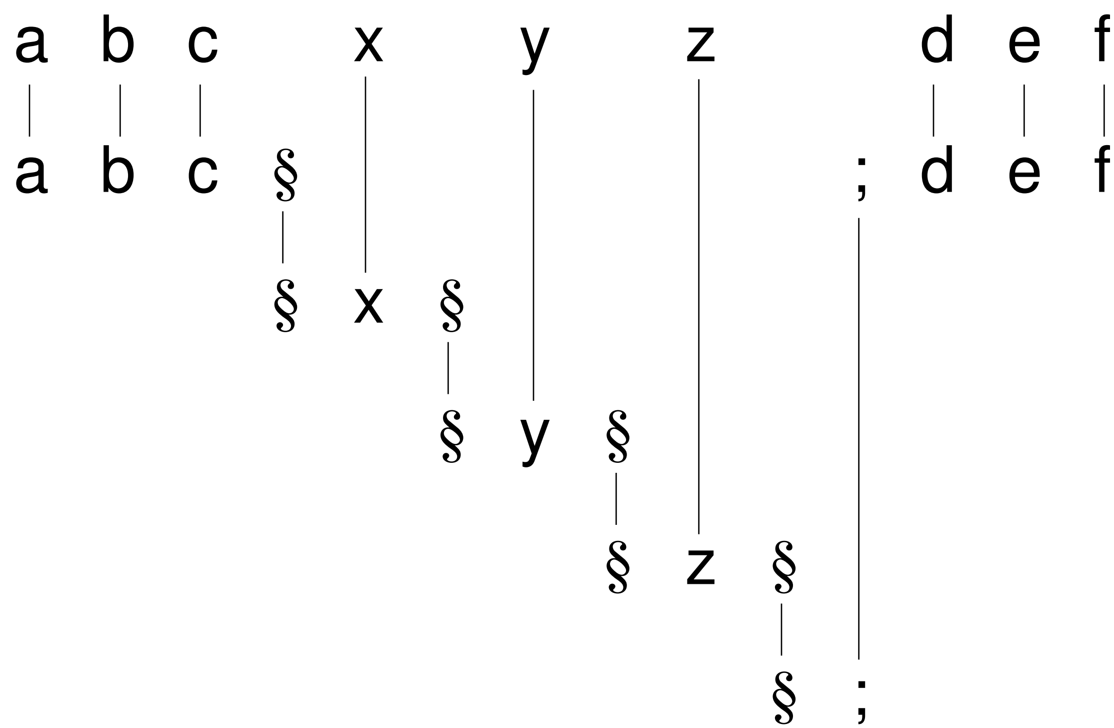


Fig. 6

Alignment:

g	(				1		)
g	(	boolean	x	#		;	)
		boolean		#	1	;	

Unification:

g (boolean x **# 1 ;)**

Fig. 7

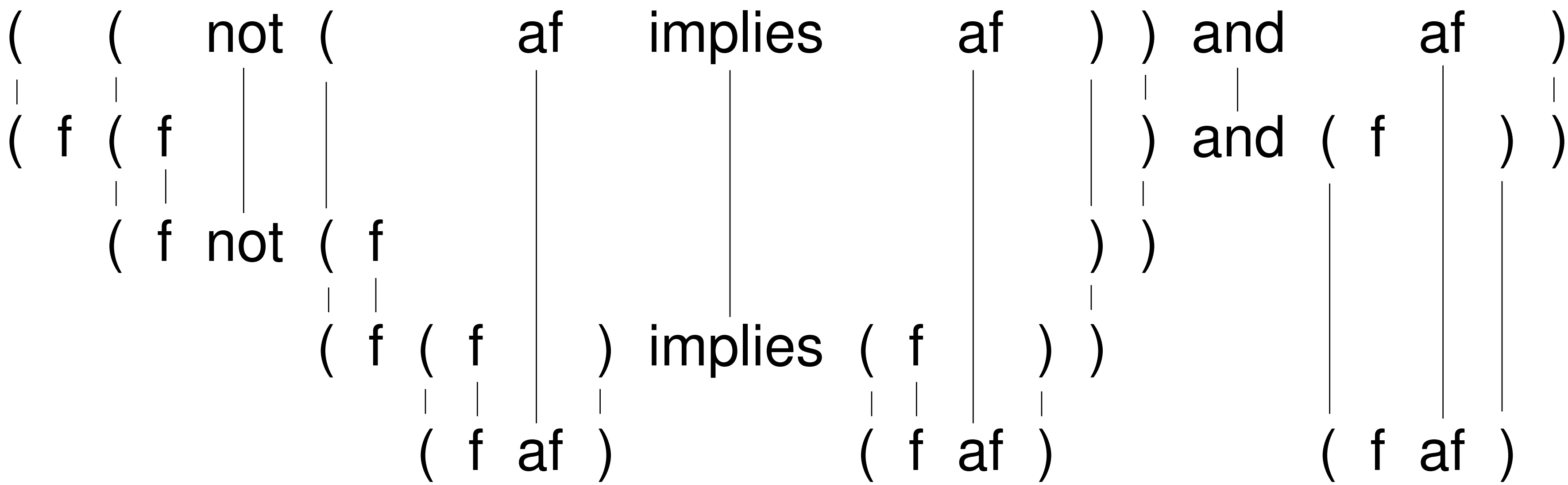
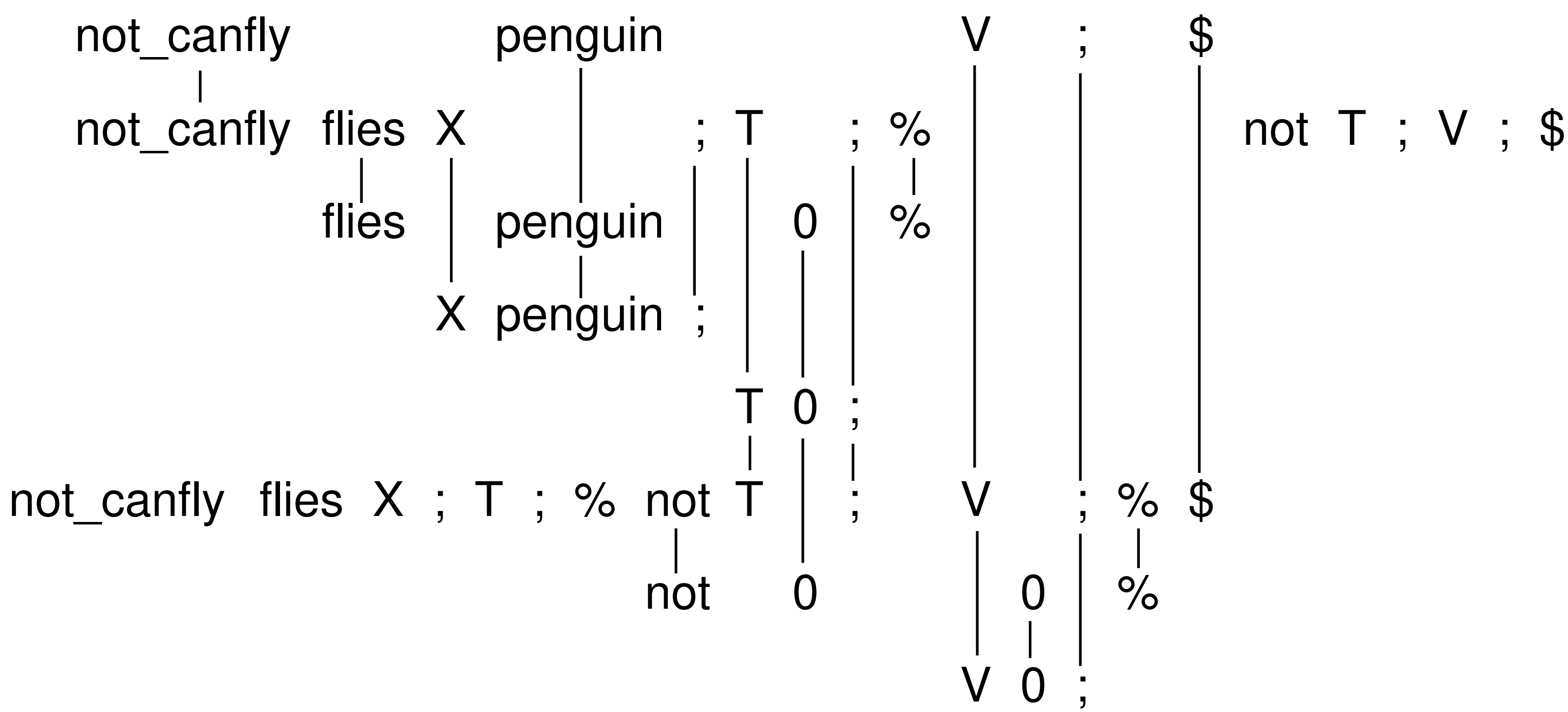


Fig. 8



not_canfly penguin V ; $
not_canfly flies X ; T ; % not T ; V ; $
flies penguin 0 %
X penguin ;
T 0 ;
not_canfly flies X ; T ; % not T ; V ; % $
not 0 0 %
V 0 ;

Fig. 9

Alignment:

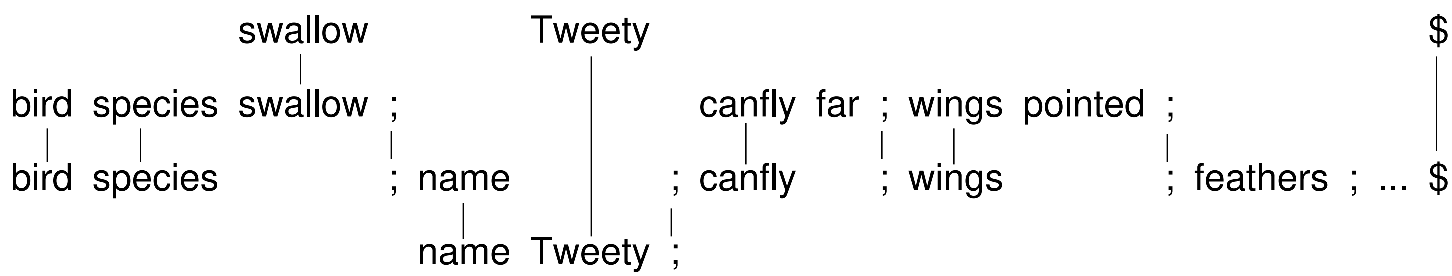
bird		Tweety									$
bird	name		;	canfly	;	wings	;	feathers	;	...	$
	name	Tweety	;								

Unification:

bird name Tweety ; canfly ; wings ; feathers ; ... **$**

Fig. 10

Alignment:



Unification:

bird species swallow ; name Tweety ; canfly far **; wings** pointed **;** feathers ; ... **$**

Fig. 11

Alignment:

```
                        Chirpy                  male                  bird      $
living_thing   name             ;  gender              ;  class               ;  $
               name     Chirpy  ;
                                   gender   male   ... ;
                                                          class   bird   ... ;
```

Unification:

living_thing **name Chirpy ; gender male** ... **; class bird** ... **;** $

Fig. 12

Alignment:

```
                  penguin         Tweety                                                  $
                     |              |                                                     |
bird  species  penguin  ;           |        canfly  no   ;  wings  stubby  ;             |
 |      |                |          |          |          |    |            |             |
bird  species           ;  name     |     ;  canfly       ;  wings          ;  feathers ; ... $
                             |      |     |
                           name  Tweety   ;
```

Unification:

bird species penguin ; name Tweety ; canfly no **; wings** stubby **;** feathers ; ... **$**

Fig. 13

Alignment:

```
                Tweety     canfly                                  $
bird  name                ; canfly  ;  wings  ;  feathers  ;  ...  $
      name  Tweety        ;
```

Unification:

bird **name Tweety ; canfly** ; wings ; feathers ; ... **$**