In *Proceedings of the First International Workshop on Inductive Logic Programming*, Portugal, 1991, pp. 177-191

ON THE INTEGRATION OF LEARNING, LOGICAL DEDUCTION AND PROBABILISTIC INDUCTIVE INFERENCE

J Gerard Wolff

School of Electronic Engineering Science, University of Wales, Bangor, LL57 1UT, UK. Telephone: +44 248 351151 ext 2691. E-mail: gerry@uk.ac.bangor.sees. Fax: +44 248 361429.

1 ABSTRACT

This paper introduces the conjecture that many kinds of cognition and computing may usefully be seen as a search for *efficiency* in information, where *efficiency* is defined in terms of Shannon's (1949) concept of *redundancy* in information.

A prototype of a new kind of computing system which is based on this theory is described in outline.

Examples from the prototype are presented showing how a search for efficiency in information may achieve autonomous inductive learning, logical deduction and probabilistic inductive inference.

2 INTRODUCTION

The nascent field of *inductive logic programming* has been defined by Muggleton (1990) as "the intersection of Logic Programming and Machine Learning". Within this field, two themes may be distinguished:

- Most attention is being paid at present to ways in which logic programs may be created by automatic or semi-automatic induction.
- Some attention has been given to the possibility that inductive learning and logical deduction may be *integrated*. A noteworthy example here is the demonstration by Muggleton & Buntine (1988) that learning of logic programs with the aid of a teacher may be achieved by a process which may be seen as an 'inversion' of Robinson's (1965) 'resolution principle' for logical deduction.

This paper is addressed mainly to the second of these two issues but the ideas which will be described are also relevant to the first. In the paper, I describe:

- A theory which we are developing - called 'SP' - which aims to integrate and simplify a wide range of concepts in cognition and computing, including inductive learning, logical deduction and probabilistic inductive reasoning.
- Examples to illustrate the scope of the theory.
- A prototype of a new kind of computing system which is based on the theory.

- Examples from the prototype which illustrate the way in which inductive learning, logical and probabilistic inference may be seen as two facets of one kind of computing process.

The paper concludes with some discussion of future directions for this research.

Other publications from this programme of research include:

- Wolff (1990) which describes the SP theory and the proposed SP system at a stage before any prototype of the SP system had been developed.
- Wolff & Chipperfield (1990) which describes the SP6 prototype and its application to inductive learning and logical deduction.
- Wolff (1991). This book gives a relatively full account of the programme of research including the earlier research on the autonomous induction of grammars which is the foundation for current work.

To minimise overlap with these other publications, particularly the second, I will present the basic ideas quite briefly, leaving space for new examples and discussion.

3 THE SP THEORY

As already mentioned, the SP theory originated in research on autonomous inductive learning of grammars. The abstract view which emerged from this work is that learning may be seen as a search for 'efficiency' in a grammar, where the efficiency of a grammar is the ratio of its 'power' to its 'size'. The power of a grammar was defined in terms of its capacity to encode raw linguistic data in an economical way and 'size' was defined straightforwardly as the number of bits of information needed to describe the grammar.

More recently, it has become apparent that these principles can probably be applied to several other areas of cognition and computing. At the same time, simpler and more general definitions of power and of efficiency have been proposed. In its current form, the SP theory may be summarised as follows.

The central conjecture in the theory is that many kinds of computing and cognition may usefully be seen as a search for *efficiency* in information where the efficiency of a body of information, *I*, is defined as the ratio of the *power* of I to its *size*. The *size* of I is simply the number of bits needed to record it. *Power* is defined as *size - redundancy*, where *redundancy*, also measured in bits, is Shannon's (1949) concept of redundancy, and means information which is in some sense repeated or unnecessary. The power of I is the non-redundant information in I.

Searching for efficiency in I means searching for redundancy in I and removing it wherever possible. With certain kinds of information, additional gains in efficiency can be achieved if one is willing to sacrifice some power. Searching for redundancy means searching for patterns which match each other and merging or *unifying* repeating instances of any pattern to make one. There are usually many alternative ways in which patterns may be unified and it is necessary to use a hill-climbing technique, or something equivalent, to find a set of unifications giving a good overall gain in efficiency.

A 'pattern' is any sequence of the atomic symbols used in I, including discontinuous sequences.

The term *unification* has been chosen for the merging of identical patterns because the idea is similar to the concept of unification in logic. But the two ideas are not exactly the same; unification in SP is a simpler idea than unification in logic. In the rest of this paper I will be using the SP meaning, unless I say otherwise.

If the concept of *simplicity* is defined as the reciprocal of *size*, then searching for efficiency may be seen as a search for a favourable combination of *simplicity* and *power*.

3.1 The Potential Scope of the SP Theory

This section gives a few examples to illustrate the potential scope of the SP theory in cognition and computing:

- *Inductive learning of grammars*. The SNPR computer model of learning which I developed in earlier research embodies the SP principles and, as a by-product of its search for efficiency in knowledge, mimics many of the known features of language learning by children (Wolff, 1988). On this evidence, the SP theory has something useful to say about human cognition.
- *Neural computing and cluster analysis*. The idea of searching for minimum entropy in information, which is similar to (but not the same as) the SP idea of searching for efficiency in information, is used in some work on 'neural' computing (eg Hinton & Sejnowski, 1986) and also in some work on cluster analysis (eg Boulton & Wallace, 1970). The SP theory is a potential bridge between 'connectionist' and 'symbolic' approaches to computing.
- *Pattern recognition*. The 'recognition' of any pattern is essentially a unification (complete or partial) of the pattern to be recognised with another pattern, already known to the recognizer. This means a hill-climbing search for the best possible match, in keeping with the SP principles.
- *Access to information in computer memory*. The CPU of a conventional computer locates information to be retrieved from memory by sending a bit pattern to the memory. Logic circuits in the memory find the section of memory which has that address. The information which is stored there can then be sent back to the CPU.

  The process of finding the required section of memory may be seen as a crude form of hill-climbing search - to find a complete match between the bit pattern sent by the CPU and one of the memory addresses represented in the logic circuits. The bit pattern from the CPU is, in effect, unified with the address which fully matches it. Memory access may thus be seen as a search for efficiency in information, albeit in a rather limited form.
- *Query by example*. Retrieving information from a database using the technique of 'query by example' means matching the query against records in the database and unifying the query with the records which best match it.
- *De-referencing of identifiers*. A familiar feature of computer systems is the use of an identifier (name) to refer to a larger body of information stored elsewhere. Examples include the use of a function name as a 'call' to a function, the name of a class (in object-oriented systems) as a reference to the corresponding class definition and the name of a file as a reference to the information stored in the file.

  In the same way that an address sent to memory by a CPU may be seen as a pattern to be unified with an identical pattern found elsewhere, the de-referencing of identifiers may be understood in terms of matching and unification. The process of searching for a 'full' match between an identifier and the corresponding label on its referent may be seen as a simple form of hill-climbing search.

- *Unification in logic*. A large part of unification as it is understood in logic is the comparison of patterns and the merging (unification in the SP sense) of patterns which are the same. Hence the conjecture that logic may usefully be interpreted in terms of the more general notion of searching for efficiency in information.

  The potentially wide scope of unification in the sense of logic is recognised in a useful review by Knight (1989).

4 SP6

SP6 is a prototype of a new kind of computing system based on the SP theory. This prototype is a software simulation of the proposed SP system, written in C and running on a Sun workstation. In the future, the SP system may be developed as a software virtual machine running on existing high-parallel hardware (eg the AMT DAP or a multi-Transputer system) or new high-parallel hardware may be developed, dedicated to the processes in the SP system.

4.1 The SP Language

In the system it is intended that all kinds of knowledge - 'data', 'programs', 'rules', 'specifications', 'models' etc - will be expressed in one simple but powerful language. A formal definition of the syntax of the SP language is given in Wolff (1990, 1991). Informally, the language is built of 'objects' of four types:

- An *Ordered-AND-object* (OAO) is a sequence of objects bounded by round brackets ('(', ')'). An OAO is similar to a list in Lisp but lacks the internal head-tail structure.
- An *Unordered-AND-object* (UAO) is an unordered collection of objects, essentially the same idea as a 'bag' in logic. UAOs are shown with square brackets ('[', ']').
- An *OR-object* (ORO) is similar to a UAO but its constituents represent *alternatives* in a given context. An ORO is distinguished by the use of curly brackets ('{', '}'). There is reason to believe that OROs are unnecessary. Our recent work suggests that the notion of alternatives may more properly be handled in the system's search for efficiency in information.
- *Simple* objects come in two sub-types:
  - A *symbol* is a sequence of one or more alphanumeric characters and is treated as an atomic unit in processing.
  - A *variable* is marked by an underscore ('_') and is similar to the concept of an unnamed variable in Prolog.

In keeping with the spirit of 'simplicity and power', the language has been designed to do as much as possible with as few constructs as possible. To this end, there is no explicit concept of a named variable. The effect of naming may be achieved by joining an unnamed variable with a symbol (or larger grouping) inside an OAO or UAO, eg (Mike _). Notice that capital letters have no special significance in the SP language, unlike 'Edinburgh' Prolog and its descendants.

We are using the term 'corpus' for all the knowledge within an SP system which is under the control of the user.

Every symbol within an SP corpus has a frequency value attached to it representing the number of 'instances' of that pattern the given symbol represents. For the sake of readability, these are not shown explicitly in the examples presented later but, to assist understanding of the workings of SP6, the symbols which are the result of new unifications are shown in bold type.

4.1.1 The Expressive Power of the SP Language

Although the syntax of the SP language is simple and the language is in many ways like a context-free phrase-structure grammar, the processes of pattern matching, unification and search which are being developed for the language means that, potentially, it has the expressive power of an augmented CF-PSG or Definite Clause Grammar. The novel way in which discontinuous dependencies in syntax may be represented in the language is described in Wolff (1990 , 1991).

4.2 Processing in SP6

In line with the principles of the SP theory, the SP6 prototype has been designed to search for redundancy in the information which is supplied to it and to remove it wherever possible. In this version of SP, no attempt has been made to raise efficiency by sacrificing power (a possibility which was mentioned earlier). SP6 works in many ways like a 'lossless' compression algorithm but with some sophistication in its search which is lacking in most ordinary systems for data compression.

The workings of SP6 should be reasonably clear from the examples (and accompanying explanation) given below. A fuller and more formal description is given in Wolff (1991).

4.3 SP, Information and the 'World'

In designing the SP system, it is intended that 'raw' data or information from the 'world' may be represented in a simple and direct way within the system using OAOs. Any stream of information from the world may be captured as a corresponding OAO within the system using the binary atomic symbols '0' and '1' (or any other appropriate set of symbols).

At present, OAOs in the SP language are one-dimensional sequences, but, in future versions of the system, it is intended that the OAO concept will be broadened to include two-dimensional 'maps' or 'pictures' and, possibly, higher dimensions too.

Similar principles apply when atomic symbols represent larger 'chunks' of data. If object A is to the left of object B in (a view of) the world, then this may be represented directly in SP as (A B) rather than left_of(A, B) or right_of(B, A) as would be done in more conventional systems. Likewise, the familiar order of days of the week may be represented as (Sun, Mon, Tues, Wed, Thurs, Fri, Sat) rather than a set of structures like follows(Mon, Sun), follows(Tues, Wed) etc or some other more or less indirect kind of representation.

4.3.1 'Information' and 'Knowledge' in SP

To say that raw information from the world may be represented in the SP language in a direct way is not to say that 'coded' forms of representation are excluded. 'Information' may be converted into 'knowledge' in SP through the processes for extracting redundancy and data compression which are at the heart of SP's search for efficiency in information. Examples will be given below.

The idea that the world may be represented as 'pictures' in SP is not in conflict either with the use of natural language or other human coding systems within SP knowledge structures. SP is intended not to recognise any distinctions amongst kinds of information which it receives. Natural language may be stored, compressed and used within the SP system in the same way as non-linguistic information. We expect the mature SP system to be an effective vehicle for integrating syntactic knowledge with semantic knowledge and for the understanding and production of natural languages.

5 AUTONOMOUS INDUCTIVE LEARNING

In this section, and the ones which follow, I discuss the possible applications of the SP theory to autonomous inductive learning, logical deduction and probabilistic inductive inference, giving examples from the SP6 prototype. Quote marks are used when concepts from these fields are first introduced because their interpretation in SP terms is still tentative. For convenience, the quote marks are dropped in later discussion.

Since the SP system is regarded as a concrete expression of the SP theory, there will be no sharp distinction in the discussion between the disembodied theory and the more abstract aspects of its realisation in a working system.

The success of the SNPR model (Wolff, 1988) in the inductive learning of grammars (without the aid of negative samples or a teacher) demonstrates the relevance of the SP principles to inductive learning.

SP6 incorporates much of this thinking in a revised framework with new objectives. Like SNPR, SP6 integrates the discovery of segmental (OAO) and disjunctive (ORO) structures but, because it has been deliberately designed to preserve ‘power’ in its knowledge, it lacks SNPR’s ability to make generalizations and correct overgeneralizations. (Designing the SP system so that it may make gains in efficiency by deliberate loss of power in its knowledge structures is something for the future.)

Examples are given in Wolff (1991) demonstrating the capabilities of SP6 in:

- Grammar discovery, including the discovery of ‘word’ segments, ‘phrase’ segments, disjunctive classes of structure, and the integration of segmental structure with class structure.
- The discovery of class hierarchies with ‘inheritance’ of attributes, in the manner of object-oriented design.

In this paper I give a new example to illustrate the potential of the SP concepts in automatic induction of grammars. The corpus for this example was prepared using this grammar:

```
S -> NP V NP
NP -> D N
D -> a | the
N -> boy | girl
V -> meets | likes
```

All 32 possible ‘sentences’ were created as OAOs like this:

```
[
    (a boy meets a girl)
    (the girl likes the boy)
    ...
]
```

The ellipsis (‘...’) indicates where that other sentences would go in a full representation of the corpus.

Because, as noted above, SP6 has no ability to generalise, it needs all 32 possible sentences in the corpus if it is to retrieve the 'correct' grammar for that corpus.

Here is the result of processing by SP6:

([%1 ({**the a**}{**boy girl**})]{**likes meets**} [%1 _])

The similarity between this final version of the corpus and the grammar which governed the construction of the original corpus should be obvious.

Because the 'noun phrase' structure repeats within the scope of one OAO, its first occurrence has been marked by SP6 with a system-generated 'tag' ('%1'). This has essentially the same meaning as 'NP' in the original grammar. The second noun phrase within the structure has been marked with the same tag together with an SP variable to show that the body of the noun phrase has been omitted in that location. This device, together with the use of OROs, is the way SP6 discards redundant information in the corpus whilst preserving non-redundant information.

6 LOGICAL DEDUCTION

In this section I discuss a selection of concepts from the world of logical deduction and the way these ideas may be understood within the SP framework.

6.1 '*TRUE*' and '*FALSE*' in SP

There are no explicit concepts of 'TRUE' and 'FALSE' in the SP theory. Although these are not primitive concepts in SP, they may be recognised in the system in much the same way as they are understood in systems like Prolog. If a particular knowledge structure is found within an SP corpus then, within the 'closed world' of the system, it is true. Correspondingly, if the structure is not found within the corpus then it is false. As with Prolog, these ideas of truth and falsehood are quite independent of what is true in the external world.

The mechanisms in SP for comparing patterns and unifying the ones which are the same can be used to check whether a given SP object is present within a given corpus. The object to be checked must be added to the corpus. If it then unifies completely with an object which is already there then the new knowledge structure is true in terms of the pre-established knowledge and false otherwise.

If unification in SP is a measure of truth, then partial unification - which is a prominent feature of how SP works - should mean degrees of truth and falsehood in the system. This is unconventional logic but it accords with everyday notions of truth.

6.2 'Negation' in SP

There is probably a case for adding a 'negation' operator at some stage as a primitive within the SP system. In keeping with the notions of TRUE and FALSE in SP as they have just been described, such an operator would have the effect of either notionally deleting the object to which it applies or inhibiting unification between that object and any other.

One argument for including a negation operator is that, otherwise, there is no sensible way of representing the everyday concept of negation within the system. Within SP, the negation of a negation (eg ~(~(TRUE))) should have the same value as the bare object without negation (TRUE).

A second argument is that a negation operator of the kind which has been described would be useful in the search for efficiency in information which is the basis of the SP theory. In the SP system, as in our everyday use of language, the use of a negative can be a powerful aid to economy. For example, given that a group of people is already known to the system (eg the village football team), it is more economical to describe another group of, say, 10 people as [Rangers [~ Fred]] than listing the members of this second group explicitly as [Peter Harry Mike ...].

6.3 'Modus Ponens'

A '*modus ponens*' form of logical argument like this:

1 *If this piece of blue litmus paper is placed in acid, then this piece of blue litmus paper will turn red.*
2 *This piece of blue litmus paper has been placed in acid.*
3 *Therefore, it will turn red.*

(example from Copi, 1986) may be represented in logical notation like this:

1 $p \supset q$
2 p
3 $\therefore q$.

The strict sense of $p \supset q$ cannot be expressed in current versions of SP because it is equivalent to $\sim(p \wedge \sim q)$ and SP has no negation operator yet.

But if the syllogism is interpreted in a more relaxed way then the essentials of it can be captured in SP6. Proposition 1 may be represented like this:

```
[
        ...
        (([litmus blue] acid)[litmus red])
        ...
]
```

This SP 'proposition' has been embedded in a larger UAO representing the complete corpus of knowledge of which such a proposition would normally be just a part. The association between litmus paper and blue colour is shown in a UAO. The placing of the blue litmus paper in acid is shown by bracketing that object with 'acid' within an OAO. The association between litmus paper and red colour is shown in another UAO. And the overall 'if ... then' sequence of events - a recurring pattern in the world - is shown by bracketing all structures within a OAO.

'Assertion' of proposition 2 may be achieved by adding it to the corpus:

```
[
        ([litmus blue] acid)
        ...
        (([litmus blue] acid)[litmus red])
        ...
]
```

If this structure is processed by the SP system, it finds the obvious redundancy within the structure and gives this result:

[
...
(([**litmus blue**] **acid**)[litmus red])
...
]

How should the result be interpreted? One way to look at it is to say that (([litmus blue] acid)[litmus red]) - which would normally be just one of a large number of propositions in a corpus - has been 'selected' by ([litmus blue] acid) and this selection should increase our confidence in the truth of the larger proposition, including the expectation that the litmus paper will turn red.

This informal view may be made more precise by using the frequency information attached to objects in the corpus. Let the frequency value for (([litmus blue] acid)[litmus red]) be, say, 510, representing the number of times this pattern has been experienced by the SP system. If the frequency value of the query is 1, then, after unification, the frequency value of ([litmus blue] acid) is 511. Consequently, the conditional probability of [litmus red] given ([litmus blue] acid) is 510/511 = 0.998 or very nearly 1. In other words, seeing blue litmus paper placed in acid would lead the system to conclude that it was nearly, but not completely, certain that the paper would turn red.

The original proposition records the fact that blue litmus paper turning red in acid has been a common occurrence in the system's past experience. Given the principle of inductive reasoning - that patterns frequently observed in the past are likely to repeat in the future - such a record may be read as a conditional statement: "If part of this pattern occurs in the future then the rest of the pattern is likely to come with it." The conditional expectation becomes a real expectation when the second proposition is added to the corpus.

Logical deduction and inductive reasoning are two forms of reasoning which are normally considered to be totally distinct. Notice how, in the SP world, they may be seen as two facets of one idea: searching for efficiency in information by matching patterns and unifying patterns which are the same.

6.4 'Modus Tollens'

The *modus tollens* form of syllogistic reasoning -

1 $p \supset q$
2 $\sim q$
3 $\therefore \sim p$

- can also be illustrated with the behaviour of litmus paper:

1 *If this piece of blue litmus paper is placed in acid, then this piece of blue litmus paper will turn red.*
2 *This piece of blue litmus paper has not turned red.*
3 *Therefore, it was not placed in acid.*

As with *modus ponens*, the use of negatives rules out a strict interpretation within the current version of SP. The first proposition may be represented in the same way as before. The fact of the blue litmus paper not turning red should be represented by the *absence* within the corpus of any object showing the litmus paper red. Since there is no added 'proposition', there will be no resulting unification and no corresponding implication that the litmus paper was placed in acid. In this slightly back-to-front way, the SP system can accommodate this kind of logical inference.

6.5 'Quantification' and 'Skolemization'

In this section I describe how syllogistic reasoning like this:

1 *All swans are white.*
2 *This is a swan.*
3 *Therefore, this swan is white.*

may be performed in SP.

The syllogism may be represented in logical notation as:

$$(\forall x \in \text{swan} \,.\, \text{white}(x) \wedge \exists y \,.\, \text{swan}(y)) \supset \text{white}(y).$$

To represent proposition 1 in SP requires some means of expressing the meaning of *all swans*. It seems that this may be achieved, in the manner of Skolemization, by bracketing 'swan' with an SP variable within a UAO like this: [swan _]. If 'swan' unifies with another instance of that symbol and the variable unifies with any number (zero or more) of other objects in the UAO, then the object [swan _] will unify with any object identified as a swan. Hence, [swan _] may be read as "any swan" or "all swans".

As it happens, the variable in SP6 will unify with only one object and so it does not exactly meet the present need. But in future versions of SP, we intend that the variable will be generalized so that it will unify with zero or more objects.

Given the more general meaning of the variable, *All swans are white* may be represented as:

[[swan _] white].

Turning to proposition 2, the existence of a particular swan may be represented, in the manner of Skolemization, like this:

[OBJECT-1 swan].

Propositions 1 and 2 may thus be represented like this:

```
[
    [[swan _] white]
    [OBJECT-1 swan]
]
```

SP6 knows only about extracting redundancy from knowledge; it does not know about logic. It 'sees' the redundancy in the first object - the fact that 'swan' occurs twice - and removes it, giving this result:

[[**swan** OBJECT-1] white]

This object may be interpreted as "The object previously referred to is a swan and it is white". This is the desired inference together with just one of the original premises. A better result would probably be [[**swan** OBJECT-1 _] white] which may be interpreted as "All swans, including the object previously referred to, are white". In developing future versions of the SP system, we will aim for this kind of result.

6.6 ‘Bag’, ‘Set’, ‘Union’ and ‘Intersection’

The UAO and ORO constructs in SP are, as mentioned earlier, essentially the same as the logicians concept of a ‘bag’ - a grouping in which items may be repeated. This is itself similar to a ‘set’ - a grouping in which constituent items may not be repeated.

If constituent items within a UAO (or ORO)are repeated, SP6 will merge the repeated items:

[A B B C] => [A **B** C]

Generally speaking, SP6 will convert all its ‘bags’ into ‘sets’. (In this and later examples, the symbols ‘=>’ is not part of the SP language but shows how the SP system changes one object into another.)

If two (or more) OROs are bracketed together within another ORO, SP6 merges them to form one ORO, unifying identical objects within them:

{{A B C D E}{C D E F G}} => {A B F G **C D E**}.

In effect, SP6 forms the ‘union’ of two (or more) sets. (SP6 should be able to do the same with UAOs but there is a weakness in the model which prevents this.)

In effect, SP6 also forms the ‘intersection’ of the two sets. The intersection of the sets is the set of objects which have been newly created by unification: C, D and E in this example. In all cases, including more complicated cases where the union of three or more sets has been formed, the system ‘knows’ how unified objects were derived and thus knows, in effect, all the several intersections which have been formed. The system cannot display this information at present but could be modified to do so if that were needed.

6.7 ‘Chains’ of Inference

In systems like Prolog, the more complicated kind of inference is made via a ‘chain’ of more basic inferences. Given a Prolog database like this:

```
route(york, peterborough).
route(edinburgh, newcastle).
route(peterborough, london).
route(newcastle, york).
route(X, Z) :- route(X, Y), route(Y, Z).
```

a query like

```
?- route(edinburgh, london).
```

(which may be read as "Is there a route between Edinburgh and London?") is answered via the ‘links’ in the chain represented by the individual facts in the database.

A corresponding SP6 database looks like this:

```
[
    [route(_ _ York, Peterborough _)]
    [route(Edinburgh Newcastle _ _ _)]
    [route(_ _ _ Peterborough London)]
    [route(_ Newcastle York _ _)]
]
```

Because, as noted above, the SP variable currently means "one unspecified object and only one", it is necessary to use multiple instances of the variable to show two or more unspecified towns on a route. As previously noted, we intend that the variable will mean "zero or more unspecified objects" in future versions of SP. In that case, only one variable will be needed in any one context.

This SP database looks like the corresponding Prolog database (without the recursive rule) but SP's 'style' is different. Given the processing which SP6 applies to all its knowledge, the SP database does not survive in that form. The redundancy in the database is extracted giving this result:

[route(Edinburgh **Newcastle York Peterborough** London)]

What SP6 has done is to work out immediately that there is a route between Edinburgh and London and arrange the towns in between in their correct order on the route. This accords with what people naturally tend to do given the same raw facts.

An SP6 'query' corresponding to the Prolog query may be constructed like this:

[route(Edinburgh _ _ _ London)].

As before, we intend that in future versions of SP only one variable would be needed in this example. The 'query' unifies completely with knowledge already in the database confirming that there is a route between Edinburgh and London.

Any SP system intended for practical use should be designed so that a knowledge base may be re-set to its original state after any query. Otherwise, the knowledge represented by the query, which may be false information, will become a part of the stored knowledge.

6.8 Transitive, Intransitive and Non-transitive Relations

Three kinds of logical relations are commonly distinguished:

- *Transitive* relations are those like 'route', just discussed. If there is a route from A to B and a route from B to C, then we can infer that there is a route from A to C.
- *Intransitive* relations are those like 'mother of'. If A is the mother of B and B is the mother of C, then A *cannot* be the mother of C.
- *Non-transitive* relations are those like 'loves'. If A loves B and B loves C we cannot infer that A loves C but we cannot rule it out either.

The 'route' example given above shows how SP6 can accommodate transitive relations. Largely because of weaknesses in its search method, SP6 does not give a satisfactory treatment of the other two kinds of relation. But it is reasonably clear how they may be accommodated within the SP theory and how they may be treated in future versions of the SP system.

What makes 'mother of' intransitive is that, in the world as we know it, any one child only ever has one mother. This fact, and the associated fact that any one mother can have one or more children, may be represented in the SP language like this:

[[mother _][%1 [child _][%1 _]]]

The object [mother _] is a 'schema' for any one mother - her name should be supplied in the variable slot. The object [%1 [child _][%1 _]] is a recursive structure for one or more children, each with a slot for his or her name.

To reveal the intransitive nature of this relation, two capabilities are needed in future versions of the SP system which are lacking in SP6:

- Given a recursive structure like this, the system should be able to make appropriate unifications. The system should be able to detect that [%1 _] will unify with [%1 [child _][%1 _]].
- The system should be able to find a set of *alternative* sets of unifications, each set of unifications with a value for the gain in efficiency which it represents.

Given an SP system with these capabilities and given structures which imply that a given child has, say, two mothers, the system should show that one or other of these mothers is the 'true' mother but not both of them.

Finally in this section, a 'non-transitive' relation between a 'lover' and the person who is 'loved' may be shown like this:

```
[[%1 [lover _][%1 _]][%2 [loved _][%2 _]]]
```

In this case, neither side of this relationship is limited in number. Consequently, there is no constraint on unifications as there is with the intransitive case.

7 PROBABILISTIC INDUCTIVE INFERENCE

The example given above of how the SP system may perform '*modus ponens*' reasoning illustrates the way in which SP blurs the supposedly sharp edges of logical deduction.

In the SP theory, most computing has a measure of uncertainty because the search space of possible unifications is normally too big for an exhaustive search to be practical; where there is recursion the search space is infinitely large and exhaustive searching is then impossible. Since, in these cases, the space of possible unifications is not fully searched, there is always the possibility with any given search that a more thorough search would give a different result. There is a possible link here with the uncertainty in formal systems implied by Godel's theorem and by 'algorithmic information theory' (eg Chaitin, 1987).

In SP, a 'logical deduction' is not qualitatively different from other kinds of inference. It is an inference, like the one shown in section 6.3, which can be made with a high level of confidence.

Here, I give another example from SP6 showing how less certain inferences may be made:

```
[
        [cloud_cover [black [cold _]]]
        [cloud_cover
                {[black
                        {[warm {rain #}]
                        [cold {snow # hail}]}]
                [white dry]
                [clear
                        {[warm {# drought}]
                        [cold {frost #}]}]}]
        [March [wind {strong light}]]
        [{June July August} warm]
        [{October November} fog]
]
```

```
=>
[
        [cloud_cover
                {[black
                        {[warm {rain #}]
                        [cold {snow # hail}]}]
                [white dry]
                [clear
                        {[warm {# drought}]
                        [cold {frost #}]}]}]
        [March [wind {strong light}]]
        [{June July August} warm]
        [{October November} fog]
]
```

The symbol '#' in the example stands for the null alternative within sets of alternatives.

The example shows a very incomplete and rather inaccurate knowledge base of meteorological expertise together with a query,

    [cloud_cover[black[cold _]]],

which may be read as "What should we expect if there are black clouds and it is cold?" The answer which is 'selected' by the system's best unification is that hail or snow are possibilities, or there need not be any precipitation at all.

As with the example in section 6.3, conditional probabilities may be calculated from the frequency values in the knowledge base. If the frequency of 'cloud_cover' is, say, 1000, the frequency of 'black' is 200 and the frequency, in that context, of 'cold' is 80, then the joint frequency of 'black cloud_cover' and 'cold' temperature is 81 (including the conjectural grouping represented by the query). If, for the given context, the frequencies of snow, no precipitation and hail are 40, 30 and 10, respectively, then the corresponding conditional probabilities of these three possibilities are 0.49, 0.37 and 0.12.

8 CONCLUSION

In this paper, I have argued that autonomous inductive learning, logical deduction and probabilistic inductive inference may all be understood as a search for efficiency information.

The SP6 model which has been described demonstrates the potential of this conjecture but there are issues still to be resolved. Current work aims to clarify these issues and capture new insights in a new and improved SP simulation:

- The search method in SP6 produces only on 'best' answer for any one corpus. Current work aims to develop a search method which will produce a set of alternative sets of unifications, graded by the gains in efficiency they represent. This is needed for all of the many problems for which there is more than one valid answer. Given this kind of search, it may then be sensible to drop the ORO construct in the SP language.

- The search method in SP6 is only sensitive to patterns made up of contiguous groupings of symbols. In future versions of the SP system we aim to develop search methods which can detect fragmented or 'discontinuous' patterns. This is needed mainly for applications in natural language processing.
- The SP variable should unify with zero or more other objects rather than just one as is the case in SP6. The number of such objects in any one context should be governed by the search for efficiency rather than being set to an arbitrary value.
- At some stage, a negation operator should be added to the system as described earlier.

9 ACKNOWLEDGEMENT

I am grateful to Paul Mather for assistance in developing the SP6 model and discussions of the ideas.

10 REFERENCES

Boulton D M & Wallace C S (1970). A program for numerical classification. *Computer Journal 13*, 63-69.

Chaitin G J (1987). *Algorithmic Information Theory*. Cambridge: Cambridge University Press.

Copi I M (1986). *Introduction to Logic*. London: Collier Macmillan.

Hinton G E & Sejnowski T J (1986). Learning and relearning in Boltzmann machines. Chapter 7 in D E Rumelhart and J L McClelland (Eds), *Parallel Distributed Processing, Vol I*, Cambridge Mass.: MIT Press, pp 282-317.

Knight K (1989). Unification: a multidisciplinary survey. *ACM Computing Surveys 21(1)*, 93-124.

Muggleton S H & Buntine W (1988). Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, San Mateo: Morgan Kaufmann, 339-352.

Muggleton S H (1990). Inductive logic programming. Invited paper for the First Conference on Algorithmic Learning Theory, Japan. Tokyo: Ohmsha.

Robinson J A (1965). A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery 12(1)*, 23-41.

Shannon C E & Weaver W (1949). *The Mathematical Theory of Communication*. Urbana: University of Illinois Press.

Wolff J G (1988). Learning syntax and meanings through optimization and distributional analysis. In Y Levy, I M Schlesinger and M D S Braine (Eds), *Categories and Processes in Language Acquisition*, Hillsdale, NJ: Lawrence Erlbaum.

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

Wolff J G & Chipperfield A J (1990). Unifying computing: inductive learning and logic. In the Proceeding of Expert Systems 90: T R Addis and R M Muir (Eds), *Research and Development in Expert Systems VII*, Cambridge: Cambridge University Press, pp 263-276.

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