

The Origins of Inductive Logic Programming: A Prehistoric Tale

Claude Sammut

Department of Artificial Intelligence
School of Computer Science and Engineering
University of New South Wales
Sydney Australia 2052
claude@cse.unsw.edu.au

Abstract

This paper traces the development of the main ideas that have led to the present state of knowledge in Inductive Logic Programming. The story begins with research in psychology on the subject of human concept learning. Results from this research influenced early efforts in Artificial Intelligence which combined with the formal methods of inductive inference to evolve into the present discipline of Inductive Logic Programming.

INTRODUCTION

Inductive Logic Programming is often considered to be a young discipline. However, it has its roots in research dating back nearly 40 years. This paper traces the development of ideas beginning in psychology and the effect they had on concept learning research in Artificial Intelligence. Independent of any requirement for a psychological basis, formal methods of inductive inference were developed. These separate streams eventually gave rise to Inductive Logic Programming.

This account is not entirely unbiased. More attention is given to the work of those researchers who most influenced my own interest in machine learning. Being a retrospective paper, I do not attempt to describe recent developments in ILP. This account only includes research prior to 1991 the year in which the term Inductive Logic Programming was first used (Muggleton, 1991). This is the reason for the subtitle 'A Prehistoric Tale'. The major headings in the paper are taken from the names of periods in the evolution of life on Earth.

ARCHÆOZOIC (PRE-LIFE)

In 1956, Bruner, Goodnow and Austin published their book *A Study of Thinking*. This was a landmark in psychology and would later have a major impact on machine learning. The book marked a reaction to behaviourism which dominated psychology for many years. Bruner and his colleagues emphasised cognitive processes and were particularly interested in the information processing model of the brain. The focus for their experiments was the human ability to categorise observations. In their preface they state:

We begin with what seems a paradox. The world of experience of any normal man is composed of a tremendous array of discriminably different objects, events, people, impressions...But were we to utilize fully our capacity for registering the differences in things and to respond to each event encountered as unique, we would soon be overwhelmed by the complexity of our environment... The resolution of this seeming paradox ... is achieved by man's capacity to categorize. To categorise is to render discriminably different things equivalent, to group objects and events and people around us into classes... The process of categorizing involves ... an act of invention... If we have learned the class "house" as a concept, new exemplars can be readily recognised. The category becomes a tool for further use. The learning and utilization of categories represents one of the most elementary and general forms of cognition by which man adjusts to his environment.

The experiments reported by Bruner, Goodnow and Austin were directed towards understanding a human's ability to categorise and how categories are learned. The results of the experiments were reported as decision-making strategies which, today, we would call algorithms.

The first question that they had to deal with was that of representation: what is a concept? They assumed that objects and events could be described by a set of attributes and were concerned with how inferences could be drawn from attributes to class membership. Categories were considered to be of three types: conjunctive, disjunctive and relational.

...when one learns to categorise a subset of events in a certain way, one is doing more than simply learning to recognise instances encountered. One is also learning a rule that may be applied to new instances. The concept or category is basically, this "rule of grouping" and it is such rules that one constructs in forming and attaining concepts.

Bruner, Goodnow and Austin were also interested in the cost of concept learning. In particular, they noted an increase in strain on subjects as the number of attributes increased. They also noted methods used to deal with

this strain. One was by the reduction of the number of attributes considered, requiring a ranking on the significance of an attribute. The other way of reducing the strain of many attributes was to combine or recode attributes into *attribute configurations*. Today, we would call this constructive induction.

One of the strategies which Bruner, Goodnow and Austin identified for learning conjunctive concepts was the *conservative focussing strategy*. "In brief, this may be described as finding a positive instance to use as a focus, then making a sequence of choices each of which alters but one attribute value of the focus ... and testing to see whether the change yields a positive or negative instance. Those attribute values of the focus ... which, when changed, still yield positive instances are *not* part of the concept. Those attribute values of the focus ... which, when changed, yield negative instances *are* part of the concept."

Whereas learning a conjunctive concept focussed on a positive instance, Bruner, Goodnow and Austin found that all appropriate strategies for learning disjunctive concepts depended upon the use of one or more negative instances. "In the most efficient strategy, one finds a negative instance and chooses instances that differ from it in a single value. If the change results in another negative instance, (given two-valued attributes), the attribute in question is irrelevant. If the change yields a positive instance, the new value is one of those that define the concept. After one value has been located, *one starts again from a negative instance*, changing the values of other attributes until, in this way, one has located all the disjunctive defining attributes". This procedure is called *negative focussing*.

PROTEROZOIC (FIRST LIFE)

This period spans the 1960's and early 70's. It represents the emergence of the key computational ideas that would lead to the development of ILP.

Banerji

In the early 1960's there was no discipline called 'machine learning'. Instead, learning was considered to be part of 'pattern recognition', which had not yet split from AI. One of the main problems addressed at that time was how to represent patterns so that they could be recognised easily. Symbolic description languages were developed to be expressive and learnable.

Banerji (1960, 1962) first devised a language which he called a 'description list', which utilised an object's attributes to perform pattern recognition. Pennypacker, a masters student of Banerji at the Case Institute of

Technology, implemented the recognition procedure and also used Bruner, Goodnow and Austin's Conservative Focussing Strategy to learn conjunctive concepts (Pennypacker, 1963). This work had a direct influence on Cohen (1978), who developed algorithms for learning concepts expressed in predicate logic. We will describe Cohen's work later.

Recognising the limitations of simple attribute/value representations, Banerji (1964) introduced the use of predicate logic as a description language. Thus, Banerji was one of the earliest advocates of what would, many years later, become ILP.

We use a simple example from Banerji (1980) to illustrate his concept description language. There is a language for describing instances of a concept and another for describing concepts. Suppose we wish to represent the binary number, 10, by a left-recursive binary tree of digits '0' and '1':

[head: [head: 1; tail: nil]; tail: 0]

'head' and 'tail' are the names of attributes. Their values follow the colon. The concepts of binary digit and binary number are defined as:

$$x \in \textit{digit} \equiv x = 0 \vee x = 1$$

$$x \in \textit{num} \equiv (\textit{tail}(x) \in \textit{digit} \wedge \textit{head}(x) = \textit{nil}) \\ \vee (\textit{tail}(x) \in \textit{digit} \wedge \textit{head}(x) \in \textit{num})$$

Thus, an object belongs to a particular class or concept if it satisfies the logical expression in the body of the description. Predicates in the expression may test the membership of an object in a previously learned concept.

Banerji always emphasised the importance of a description language that could 'grow'. That is, its descriptive power should increase as new concepts are learned. This can clearly be seen in the example above. Having learned to describe binary digits, the concept of digit becomes available for use in the description of more complex concepts such as binary number. The goal of building an incremental system like this strongly influenced Brian Cohen, my thesis adviser, when he developed CONFUCIUS, and me, when I developed Marvin.

Plotkin

Plotkin's (1970) work "originated with a suggestion of R.J. Popplestone that since unification is useful in automatic deduction by the resolution method, its dual might prove helpful for induction. The dual of the most general unifier of two literals is called the least general generalisation".

At about the same time that Plotkin took up this idea, J.C. Reynolds was also developing the use of least general generalisations. Reynolds (1970)

also recognised the connection between deductive theorem proving and inductive learning:

Robinson's Unification Algorithm allows the computation of the greatest common instance of any finite set of unifiable atomic formulas. This suggests the existence of a dual operation of 'least common generalization'. It turns out that such an operation exists and can be computed by a simple algorithm.

A notion of generality and a way of computing a generalisation is the basic machinery required to build an effective symbolic learning system. Plotkin used θ -subsumption as a means of defining generalisation. θ -subsumption is a partial ordering of generality over clauses. A clause, C , θ -subsumes another clause, D , if and only if there is a substitution, θ , such that $C\theta \subseteq D$, where clauses C and D are treated as sets of literals. A least generalisation of a set of clauses is a generalisation which is less general than any other such generalisation.

This basic notion of LGG's made it possible to perform induction on a set of clauses provided as data. However, it could not take into account an existing theory. Plotkin (1971a) imagined a robot that could see, handle and touch things. It could explore its environment and form generalisations about the objects it encountered. He assumed that the robot could start with some theory of the world which would be extended through experience. Thus, any generalisations that the robot performed would have to be done relative to its current theory. Plotkin extended the LGG to the *relative least general generalisation* (RLGG).

Plotkin (1971b) defines generalisation relative to a background theory (i.e. a logic program) as follows: C generalises D relative to P if there exists a substitution θ such that $P \models \forall(C\theta \rightarrow D)$. Plotkin's method of constructing RLGGs could only be guaranteed to produce a clause of finite length if the background theory, P , was a conjunction of ground literals and even this computation was intractable.

Despite the computational inefficiencies, Plotkin's work had profound influence on researchers many years later. Unfortunately for machine learning, Plotkin himself, never returned to this subject after he completed his Ph.D. thesis.

PALÆOZOIC (ANCIENT LIFE)

In the 70's, machine learning was still emerging as a distinct discipline in AI. In this section we describe some of the work in this period that is related to ILP. As we will see there are some dead ends and promising efforts that were never pursued.

Michalski

Although Ryszard Michalski is not normally associated with Inductive Logic Programming, it should be remembered that he and his students attempted to develop learning systems in which concepts were expressed in an Augmented Predicate Calculus, as it eventually became known. The strongest attempt was the INDUCE program, which will be described presently. This work is presented here as an illustration of good intentions that could not be fulfilled because of the lack of a unifying principle that could give the work solid foundations.

Variable Valued Logic was the name originally given by Michalski (1973) to an augmented form of predicate logic. One of the main reasons for developing this class of languages was to make concept descriptions more readable for humans. So Michalski, influenced to some extent by Donald Michie, was among the first machine learning researchers to pay attention to the human use of the results of machine learning. Something that, today, we take very seriously.

To achieve human readability, according to Michalski, the number of disjunctions should be minimized, the number of predicates in a conjunction should also be kept small. Recursion must be avoided if possible.

The basic elements of VL languages are called selectors. Some examples are:

```
[colour(box1) = white]
[length(box1) >= 2]
[weight(box1) = 2..5]
[blood-type(P1) = O, A, B]
[on-top(box1, box2)]
[weight(box1) > weight(box2)]
[type(P1).type(P2) = A, B]
```

Variable Valued Logic was intended to be a general purpose description language. In order to allow a program using VL to operate in a specific domain, the user supplied the system with domain knowledge. This was done by specifying the type and range of the descriptors that would be used. In addition, the user could describe the properties of predicate functions such as

$$\forall x_1, x_2, x_3 ([\text{left}(x_1, x_2)][\text{left}(x_2, x_3)] \rightarrow [\text{left}(x_1, x_3)])$$

which states that if x_1 is left of x_2 and x_2 is left of x_3 then x_1 is left of x_3 .

While Michalski recognised the importance of background knowledge in learning, none of his learning systems could add learned knowledge into its store of background knowledge.

The learning algorithm used in the program, INDUCE-1.1 is described by Dietterich (1978) and Michalski (1980). Briefly, the program begins by augmenting the data rules input by the user, by using the inference rules in the domain knowledge to produce new rules. For example, if an example includes a triangle, the fact that it is also a polygon is also added. When all the inferred rules have been added, the program begins a beam search for most specific generalisations. Throughout the search there are a number of mechanisms to limit the branches which are pursued.

The search proceeds as follows: The algorithm builds a sequence of sets called *partial stars*, denoted by P_i . An element of a partial star is a product (i.e. conjunction of selectors). The initial partial star (P_i) consists of the set of all selectors, e_i , from the description of the training example. These are considered single element products. A new partial star P_{i+1} is formed from an existing partial star P_i such that for each product in P_i , a set of products is placed into P_{i+1} where each new product contains the selectors of the original product plus one new selector of the e_i which is not in the original product. Before a new partial star is formed, P_i is reduced according to a user defined optimality criterion to the 'best' subset before a new partial star is formed.

While Michalski's Aq family of learning algorithms had a strong influence on machine learning, INDUCE was less successful. The star method was developed for propositional learning systems and extended, in INDUCE, to first-order logic. Unfortunately, the method does not scale well, because of its computational complexity.

In creating the description language and his model of generalisation, Michalski invented many special cases. The result was a complex system in which it is difficult to find a unifying principle which the implementor can take advantage of. As we shall see later, inverse resolution operators provide all the power of Michalski's generalisation system, but with much greater simplicity.

Vere

Steven Vere made great strides in developing a logical basis for machine learning, yet few people today recognise his efforts. Vere's work was concerned with developing formal induction algorithms for expressions in predicate calculus (Vere, 1975). He saw his work as creating a dual for deductive theorem proving, in the spirit of the earlier work of Reynolds and Plotkin and many of his ideas are derived from Plotkin (1970).

In Vere's concept description language a literal is a list of terms like $(ON .X1 .X2)$. An identifier preceded by a period is a variable. Other terms, such as, ON are constants. A *product* is a conjunction of literals.

To illustrate Vere's original generalisation algorithm for conjunctive concepts, we will use one of his own examples (Vere, 1975). We wish to find a maximal common sub-expression of,

$$\begin{aligned} R1 &= (B \ X2) \ (W \ X3) \ (C \ X2 \ R) \ (S \ X3 \ X2) \\ R2 &= (W \ X6) \ (B \ X5) \ (S \ X6 \ X5) \ (D \ X6 \ E) \end{aligned}$$

Literals from R1 and R2 which are the same length and contain at least one term in common in the same position are paired.

$$\begin{aligned} P1 &= (B \ X2) \ (B \ X5) \\ P2 &= (W \ X3) \ (W \ X6) \\ P3 &= (S \ X3 \ X2) \ (S \ X6 \ X5) \end{aligned}$$

Terms in the pairs which do not match are replaced by variables producing the generalisation,

$$(W \ .Y) \ (B \ .Z) \ (S \ .Y \ .Z)$$

In this example, only one set of pairs could be produced but some matching problems may result in several possible ways of pairing literals. In this case the pairs which give the longest (most specific) generalisation are chosen.

If the concept being learned is conjunctive, then it is sufficient to find the intersection of all the products to produce the generalisation. However, if the concept is disjunctive then it is not possible to simply match descriptions of instances. If two instances belong to different disjuncts, then the matching descriptions will produce an empty or incorrect generalisation. Therefore Vere (personal communication) adopts the following modification:

Suppose the instances shown to the program are I1, I2, I3, I4, I5, I6. The procedure begins by trying to generalise I1 and I2 to find a maximal common generalisation (mcg). Suppose also that I1 and I2 are successfully generalised into mcg G1. We then try to generalise G1 and I3. Suppose this attempt fails because I3 belongs to a different disjunct to I1 and I2. We jump over I3 and try to generalise G1 and I4. This succeeds giving mcg G2. Now we try to generalise G2 and I5. This fails. Then we try to generalise G2 and I6. This succeeds giving mcg G3. The procedure then begins another pass through the remaining instances, that is, through the list I3, I5. Suppose I3 and I5 can be successfully generalised into G4, then we obtain the disjunctive generalisation $G3 \vee G4$.

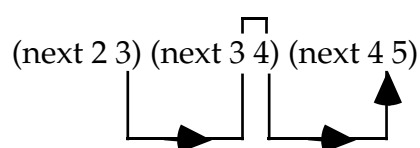
Vere (1977) developed methods for learning in the presence of background knowledge. For example, to teach the poker concept of a 'full house', it is sufficient to show the program two hands both of which have three cards with the same number and the remaining two with a different number. No information is required beyond that present in the description of the examples. However, to learn the concept 'straight', the program must

know something about the ordering of cards in a suit to recognise that the hands shown as examples contain a numerical sequence. This background information may be represented as:

(next 2 3) (next 3 4) ... (next K A)

When a hand is shown as an instance of 'straight', the program must be able associate the description with the background information. To describe how relevant background literals are selected, requires the concepts of an association and association chains.

Two literals L_1 and L_2 have an *association* if and only if the i^{th} term of L_1 is identical to the j^{th} term of L_2 . An *association chain* is a sequence of associations. For example:



Vere's algorithm looked for association chains in which the first and last literals are foreground literals (i.e. appear in the example description) and the rest are background literals. In practice, limits were placed in the lengths of association chains. In a very informal sense, this method foreshadows *ij*-determinacy used in modern ILP systems to reduce search.

Sometimes it is necessary to specify exceptions to a rule. For example, an almost universal criterion for establishing that an animal is a bird is that it flies. However, there are some exceptions. Bats are mammals, but they fly. To express this it is necessary to introduce logical negation. For example, $(\text{flies } .X) \sim (\text{bat } .X)$ could describe the concept 'bird'. Vere's THOTH program was capable of learning expressions of the form,

$$P \sim (N1 \sim (N2 \sim \dots))$$

P is a product which represents the concept. $N1$ is a product which describes an exception to P , $N2$ is an exception to the exception, etc. The negative products are called *counterfactuals*.

Taking another of Vere's examples (Vere, 1980), the task of the learning system is to find a description which discriminates between the set of objects on the right and the set on the left in Figure 1. It is not possible to produce such a description without making exceptions. The objects on the left are described as having an object X on top of an object Y . Y is a green cube. X must not be blue except if it is a pyramid.

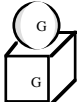
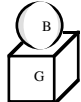
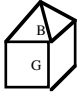
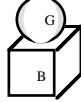
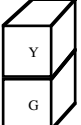
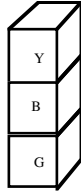
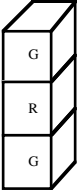
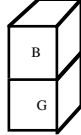
POSITIVE INSTANCES	NEGATIVE INSTANCES
<p>(ON T1 T2) (SPHERE T1) (GREEN T1) (CUBE T2) (GREEN T2)</p>  <p>ρ1</p>	<p>(ON T10 T11) (SPHERE T10) (BLUE T0) (CUBE T11) (GREEN T11)</p>  <p>v1</p>
<p>(ON T3 T4) (PYRAMID T3) (BLUE T3) (CUBE T4) (GREEN T4)</p>  <p>ρ2</p>	<p>(ON T12 T13) (SPHERE T12) (GREEN T12) (CUBE T13) (BLUE T13)</p>  <p>v2</p>
<p>(ON T5 T6) (CUBE T5) (YELLOW T5) (CUBE T6) (GREEN T6)</p>  <p>ρ3</p>	<p>(ON T14 T15) (ON T15 T16) (CUBE T14) (YELLOW T14) (CUBE T15) (BLUE T15) (CUBE T16) (GREEN T16)</p>  <p>v3</p>
<p>(ON T7 T8) (ON T8 T9) (CUBE T7) (GREEN T7) (CUBE T8) (RED T8) (CUBE T9) (GREEN T9)</p>  <p>ρ4</p>	<p>(ON T17 T18) (CUBE T17) (BLUE T17) (CUBE T18) (GREEN T18)</p>  <p>v4</p>

Figure 1. Vere's counterfactuals example.

A generalisation of the examples, ρ1, ρ2, ρ3 and ρ4 is

$$(ON .X .Y) (GREEN .Y) (CUBE .Y)$$

Without qualification, this is unacceptable since it also admits the negative examples, v1, v3 and v4. We use these exceptions as the positive examples of a residual generalisation which yields (BLUE .X). Thus the new description for the examples on the left is:

$$(ON .X .Y) (GREEN .Y) (CUBE .Y) \sim (BLUE .X)$$

This now excludes ρ2. So a new residual generalisation is formed for the exception to the exception. This yields the final description:

$$(ON .X .Y) (GREEN .Y) (CUBE .Y) \sim ((BLUE .X) \sim (PYRAMID .X))$$

Much later, Srinivasan, Muggleton and Bain (1992) would use a related method in their non-monotonic learning.

Cohen

Brian Cohen's work proceeds directly from Banerji and Pennypacker. Banerji (1969) suggested that it would be possible to create effective descriptions by learning domain knowledge. This is the approach taken by Cohen in his program, CONFUCIUS (Cohen, 1978; Cohen & Sammut,

1982) . The description language, called CODE, becomes more expressive as more knowledge is acquired.

Simple expressions are of the form:

$$\begin{aligned} \text{colour}(X) &= \text{red} \\ x &\in \text{set1} \\ \text{set1} &\supset \text{set2} \end{aligned}$$

For each operator there is also the negation, enabling the representation of exceptions. There is also another operator, *contained-in* which is true when an object is contained in a concept that is in CONFUCIUS' memory. Thus,

$$\begin{aligned} (X, Y) \text{ contained-in connected iff} \\ \text{neighbour}(X) = Y \text{ and} \\ \text{neighbour}(Y) = X \end{aligned}$$

recognises points X and Y which are connected by a line segment.

$$\begin{aligned} (X, Y, Z) \text{ contained-in triangle iff} \\ (X, Y) \text{ contained-in connected and} \\ (Y, Z) \text{ contained-in connected and} \\ (Z, X) \text{ contained-in connected} \end{aligned}$$

recognises the triangle described by the vertices X, Y and Z. Notice that *triangle* used *connected* in its description. Knowledge of triangles requires a knowledge of straight lines, as one would expect. This demonstrates the way in which CONFUCIUS learns to understand more about its world as it learns more concepts.

Disjunctive concepts can also be expressed in CODE. The language also allows recursion which is essential for describing abstract concepts of among other things, numbers and lists. The main goal influencing the design of CODE was the ability of one concept to refer to another, i.e. CODE is a growing language.

The original learning algorithm used in CONFUCIUS was derived from the work of Pennypacker (1963). The algorithm developed by Cohen (1978) for CONFUCIUS is:

1. An instance is presented to the program by the trainer.
2. The program generates all the true statements it can to describe the exemplar. This includes statements describing containment in previously learned concepts.
3. CONFUCIUS then proceeds to remove statements from the description. Remember that a subset of a description is a more general description.
4. The new hypothesis obtained by the removal is tested to see if it is more or less general than the target concept. This may be done in either of two ways:

- by showing the description of the hypothesis to the trainer and asking if it is part of the concept to be learned
- or if negative examples have been supplied, by seeing if the hypothesis recognises any of the negative instances.

In implementing this search method there is one major obstacle to overcome. Suppose the statements

$$\text{colour}(X) = \text{red} \wedge \text{colour}(Y) = \text{red} \wedge \text{colour}(X) = \text{colour}(Y)$$

are in the hypothesis. If only one of the statements is removed, then the hypothesis is no more general than it was before, because the removed statement is implied by the remaining two. Thus CONFUCIUS must be able to keep track of implications and remove sets of statements in order to generalise the concept.

CONFUCIUS was the first program to learn descriptions in first-order logic to be able to be able to reuse learned concepts. Learned concepts were stored by the program and could be used in further learning. To facilitate matching examples with stored concepts, Cohen developed a complex pattern matching system. This would later be replaced by unification in Marvin (Sammut, 1981), the successor to CONFUCIUS.

MESOZOIC (MIDDLE LIFE)

This period marks the emergence of the first programs that could claim to be performing inductive logic programming. Shapiro' MIS and my program, Marvin, were both capable of learning concepts that could be executed as logic programs.

Sammut

My own learning system, Marvin, was a direct descendent of CONFUCIUS. The main steps involved in the algorithm are:

<i>Initialise</i>	A single example presented by a trainer is described by a restricted form of First Order Predicate Logic, equivalent to Horn clauses. This description forms the first hypothesis for describing the concept to be learned.
<i>Generalise</i>	Marvin tries to generalise the hypothesis. If it is not possible to create a generalisation, the learning process stops.
<i>Test</i>	The generalisation is tested by constructing an object belonging to the hypothesised concept. The object is

shown to the trainer. If he agrees that the object is recognised by the concept to be learned then *generalise*.

Specialise If the hypothesised concept recognises objects not belonging to the concept to be learned then a new, more specific hypothesis is created. *Test* the more specific hypothesis.

A new example must be shown for each disjunct in disjunctive concept.

My thinking about Marvin evolved over time. Having developed from the world of concept learning, my original focus was on object descriptions using attributes and values. These descriptions was turned into expressions in conjunctive normal form in first order logic. Eventually, I realised that I was dealing with the equivalent of Horn clauses. So by the time details of Marvin were published (Sammut and Banerji, 1986), the program was seen as a true Inductive Logic Programming System.

The learning algorithm is a combination of generalisation and object construction procedures. The aim is to begin with an initial hypothesis for the concept, called trial T_0 , and using the generalisation procedure, create a succession of new trials T_i . Eventually there will be a T_n such that any attempt to produce a further generalisation, T_{n+1} will result in an *inconsistent generalisation*. A generalisation is said to be inconsistent if the resulting trial recognises events that the concept to be learned does not. T_n is called the *target*.

A new trial can be created in either of two ways: If the current trial, T_i is consistent, then T_{i+1} is created by applying the generalisation procedure which replaces some literals by a single, more general one. However, if T_i is inconsistent then we do not want to generalise the trial. Instead we must create a T_{i+1} which is *more specific* and does not recognise those events which the target does not. A more specific trial may be created by *adding* new literals or returning removed literals to the description.

Marvin made a number of contributions to the development of ILP.

Instance Construction

Marvin was one of the first learning programs to test its generalisations by showing the trainer instances of its hypotheses. The idea to do this in Marvin originated with my thesis adviser, Brian Cohen. His program, CONFUCIUS tested its generalisations by showing the hypothesised concept to the training. Thus, the trainer had to know the logical definition of the concept in order to answer correctly. The burden on the trainer could be eased by asking him to comment only on examples.

Instance construction necessitated executing the concept description as a program that builds objects. It became clear to us that we were now overlapping with two other areas of research, namely, logic programming and automatic programming (Cohen and Sammut, 1980). At about the same time, and unknown to us, Shapiro (1981a, b) was also making the same connections.

Generalisation

Marvin's generalisation method was used by Buntine (1986, 1988) in his theory of generalised subsumption and this formed the foundation for Muggleton and Buntine's (1988) *absorption* operator. Rouveirol's (1990) *saturation* operator is also derived from Marvin.

Assuming that concepts are stored as Horn clauses, the basis of Marvin's generalisation operation was to find a subset of literals, within the current hypothesis, that matched the body of a clause already stored in memory. The matching subset could then be replaced by a new literal constructed from the head of the clause, under the substitutions that arose from the match. In Marvin, all matches were performed before any replacement was attempted. Since Marvin tested hypotheses by constructing instances, the program had to ensure that all variables could be grounded, thus the replacement operation had to be modified to preserve this requirement.

One of the difficulties with testing hypotheses by showing examples is that you can never show enough examples to be sure that the hypothesis is correct. In Marvin, I made the assumption that all predicates that were required to describe a concept were already present in memory. I showed, that this assumption was sufficient to ensure that only one example was needed to distinguish consistent and inconsistent hypotheses. Without that assumption, Marvin could not work. This problem could be helped if Marvin could invent its own predicates.

Although it was never implemented, my Ph.D. thesis (Sammut, 1981) described what would later be called intra-construction (Sammut, 1985).

Specialisation

Few ILP systems combine both generalisation and specialisation. Marvin did not attempt to make least general generalisations. Instead, it made conservative generalisations, which might require refinement. Thus, when an inconsistent generalisation arose, Marvin employed heuristics for finding likely specialisations that would be consistent with the target concept.

Incremental

Like its predecessor, CONFUCIUS, the concepts that Marvin learned were stored in its memory, available for later use. Concepts that had been

learned previously, could take part in pattern matching while learning new concepts. Thus, these programs fulfilled Banerji's desire for learning systems that used a growing description language.

The biggest mistake I made in developing Marvin was in not seeing the connection between our approach and Plotkin's. Brian Cohen and I were aware of his work, but failed to realise that we could merge subsumption with rewriting operations to invert resolution.

Shapiro

Shapiro (Shapiro, 1981a; 1981b) was one of the first to draw on results from computational learning theory to build a machine learning program with a sound theoretical basis. Using Gold's (1967) framework of *learning in the limit*, he was able to prove that his algorithm would eventually learn a target concept. Shapiro was also the first to explicitly use a Horn clause representation and employ the resolution procedure in his learning algorithm. The problem he tried to solve can be expressed as follows (Shapiro, 1981a):

In a model inference problem we assume some unknown model M for a given first order language L . We distinguish two types of sentences in L : *observational sentences*, which correspond to descriptions of experimental results, and *hypotheses*, which can serve as explanations for these results. The model inference problem is,

Given the ability to test observational sentences for their truth in some unknown model M , find a finite set of hypotheses, true in M , that imply all true observational sentences.

The Model Inference System works as follows. The system has an initial theory, which is empty. It attempts to construct hypotheses to add to the theory to explain given examples.

If a theory is too general, i.e. it implies a false observational sentence, one can conclude that at least one of its clauses is false. Shapiro describes a *contradiction backtracing algorithm* which can detect such hypotheses by performing *crucial experiments* in the model. The theory can then be generalised by removing this false hypothesis from it.

When a false hypothesis is added to a theory it becomes possible to derive, by resolution, a false sentence. Backtracing is a procedure that works through the proof tree of this derivation to determine which hypothesis is false. The resolution tree is ordered so that the atom resolved upon appears in the condition of the left child and in the conclusion of the right child of the resolvent. The algorithm starts from the root \square , and iteratively tests the atoms resolved upon. If the atom is true in M it

chooses the left subtree, otherwise the right subtree, until it reaches a leaf. The hypothesis in the leaf is false in M and is removed from the theory.

If a theory is too specific, i.e. true observational sentences cannot be derived, then the theory is missing a hypothesis. Rather than trying to add some arbitrary new clause, we can improve the efficiency of the algorithm by refining an already refuted theory. A *refinement operator* may add new literals, replace variables by functions (which could be zero arity, i.e. constants) or by binding variables to each other. If there are no previously refuted hypotheses, the algorithm begins with the null hypothesis.

This MIS learning algorithm is shown below:

```
Set the theory  $T$  to {  $\square$  }
repeat
    Examine the next example
    repeat
        while the theory  $T$  is too general do
            Specialise it by applying
            contradiction backtracing and remove
            from  $T$  the refuted hypothesis
        while the theory is too specific do
            Generalise it by adding to  $T$ 
            refinements of previously refuted
            hypotheses
    until the conjecture  $T$  is neither too general nor too
        specific with respect to the known facts
    Output  $T$ 
forever
```

Shapiro proved that with the most general refinement operator, MIS will learn the target concept in the limit. Refinement operators are devices for searching the space of sentences in the hypothesis language. More expressive languages and more general refinement operators give rise to extremely large search spaces. Thus, from a practical standpoint, the major drawback of Shapiro's work was that it left open the question of efficient search.

CAINOZOIC (NEW LIFE)

We now come to the final stage of evolution prior to the first publications using the term *Inductive Logic Programming*. Our discussions in this section are brief since much of the work is well known to today's researchers. We only refer to some of the key papers to show how previous research has influenced current ideas.

Buntine

Wray Buntine revived interest in subsumption as a model for generalisation in his 1986 and 1988 papers. Buntine was particularly concerned to improve the notions of generalisation relative to a

background theory. The generalisation method employed in Marvin (Sammut, 1981) suggested Buntine's improved model of generalisation.

The following definition of generalisation is taken from Buntine (1988). Clause C is more general than clause D with respect to logic program P if for any Herbrand interpretation I such that P is true in I , and for any atom A , C covers A in I whenever D covers A . A clause C covers a ground atom A in interpretation I if there is a substitution θ such that $C_{head}\theta$ is identical to A and $\exists(B_{body}\theta)$ is true in interpretation I .

Buntine's work was important because it built a bridge between past research and what was to come in *inverse resolution*..

Muggleton

Muggleton's first foray in ILP was at a propositional level. His program, DUCE (Muggleton, 1987) used rewriting operators to transform a theory consisting of propositional Horn clauses into a smaller, more general theory. The operators are the propositional equivalent of what are now called inverse resolution operators. This method was, to some extent, suggested by the *absorption* operator, which was first used in Marvin (Sammut, 1981). One of the most interesting features of DUCE, was that it could invent its own predicates (or theoretical terms). The inter-construction and intra-construction operators could introduce new symbols into the description language if they produced a reduction in the size of the theory. This provided DUCE with a very powerful tool for growing its language, in the sense of Banerji (1969).

By combining DUCE's operators with the more powerful generalisation models of subsumption, it became possible to build a first-order learning system with similar properties. Muggleton teamed with Buntine (Muggleton and Buntine, 1988) to produce CIGOL. CIGOL interacts with an oracle to build logic programs from examples. It was this system that most clearly demonstrated that induction could be seen as the inverse of resolution theorem proving.

CIGOL was able to construct quite complex programs, but was limited in its efficiency. To handle large training sets Muggleton and Feng (1990), with assistance from Buntine, returned to Plotkin's relative least general generalisations to reduce the large search involved in finding generalisations. In doing so, they had to abandon predicate invention as it is not yet possible to find an efficient way of discovering relevant new predicates. This is the subject of ongoing research.

Muggleton's current research is beyond the scope of this paper. But his work crystallised the field of ILP and sparked further research.

Quinlan

Quinlan's approach to learning relations expressed as Horn clauses is unusual. Whereas most of the systems discussed in this paper are in some way related to inverting resolution, FOIL (Quinlan, 1990) uses a purely data-driven induction method. Quinlan uses a general-to-specific search based on the covering algorithm in Michalski's Aq (Michalski, 1983). The basic algorithm is:

- Find a conjunction of conditions that is satisfied by some objects in the target class, but no objects from another class;
- Append this conjunction as one disjunct of the logical expression being developed;
- Remove all objects that satisfy this conjunction and, if there are still some remaining objects of the target class, repeat the procedure.

FOIL finds conjunctions by specialising a clause. The program searches for literals to add to the clause and uses information gain, like ID3 (Quinlan, 1986), to choose among the candidate literals.

Although not directly related, FOIL solves some of the problems of Shapiro's MIS. MIS uses a general-to-specific search to refine theories, but this search is very inefficient. Quinlan has presented a method of performing such a search with greater efficiency by using ideas developed in other areas of machine learning.

CONCLUSION

There are now too many new ILP systems to mention in this paper. Here, I have tried to give some of the history of ideas that have led to the current research in Inductive Logic Programming. The story began with concept formation research in psychology which influenced early machine learning efforts. The 1960's and 1970's saw the development of the notion of using logic as a concept description language that could grow. Also, the search for an inductive dual to the deductive method of resolution began. In the 1980's, the first ILP systems began to emerge. During the 1990's we have seen this area develop into a well-founded and, at the same time, a practically useful tool.

Bibliography

- Banerji, R. B. (1960). An Information Processing Program for Object Recognition. *General Systems*, 5, 117-127.
- Banerji, R. B. (1962). The Description List of Concepts. *Communications of the Association for Computing Machinery*, 5(8), 426-432.

- Banerji, R. B. (1964). A Language for the Description of Concepts. *General Systems*, **9**, 135-141.
- Banerji, R. B. (1969). *Theory of Problem Solving - An Approach to Artificial Intelligence*. New York: American Elsevier.
- Banerji, R. B. (1980). *Artificial Intelligence: A Theoretical Approach*. New York: North Holland.
- Bergadano, F., & Giordana, A. (1988). A knowledge intensive approach to concept induction. In J. Laird (Eds.), *Proceedings of the Fifth International Conference on Machine Learning*. (pp. 305-317). Ann Arbor: Morgan Kaufmann.
- Bruner, J. S., Goodnow, J. J., & Austin, G. A. (1956). *A Study of Thinking*. New York: Wiley.
- Buntine, W. (1986). Generalised Subsumption. In *Proceedings of European Conference on Artificial Intelligence*. London.
- Buntine, W. (1988). Generalized Subsumption and its Applications to Induction and Redundancy. *Artificial Intelligence*, **36**, 149-176.
- Cohen, B. L. (1978) *A Theory of Structural Concept Formation and Pattern Recognition*. Ph.D. Thesis, Department of Computer Science, University of New South Wales.
- Cohen, B. L., & Sammut, C. A. (1980). Program Synthesis Through Concept Learning. In Y. Kodratoff (Ed.), *Proceedings of The International Workshop on Program Construction*, Bonas, France: I.N.R.I.A.
- Cohen, B. L., & Sammut, C. A. (1982). Object Recognition and Concept Learning with CONFUCIUS. *Pattern Recognition Journal*, **15**(4), 309-316.
- Emde, W. (1987). Non-cumulative learning in METAXA.3. In J. McDremott (Ed.), *Tenth International Joint Conference on Artificial Intelligence*, (pp. 208-210). Milan: Morgan Kaufmann.
- Emde, W., Habel, C. U., & Rollinger, C.-R. (1983). The discover of the equator or concept driven learning. In A. Bundy (Ed.), *Eighth International Joint Conference on Artificial Intelligence*, (pp. 455-458). Karlsruhe: Morgan Kaufmann.
- DeRaedt, L., & Bruynooghe, M. (1989). Towards friendly concept learners. In N. S. Sridharan (Ed.), *Eleventh International Joint Conference on Artificial Intelligence*, (pp. 849-856). Detroit, MI: Morgan Kaufmann.
- De Raedt, L., & Bruynooghe, M. (1992). An Overview of the Interactive Concept-Learner and Theory Revisor CLINT. In S. Muggleton (Eds.), *Inductive Logic Programming*. (pp. 163-191). Academic Press.

- Dietterich, T. (1978). *INDUCE 1.1 - The program description and a user's guide* No. Department of Computer Science, University of Illinois at Urbana-Champaign.
- Gold, E. M. (1967). Language Identification in the Limit. *Information and Control*, **10**, 447-474.
- Hayes-Roth, F. (1973). A Structural Approach to Pattern Learning and the Acquisition of Classificatory Power. In *First International Joint Conference on Pattern Recognition*. (pp. 343-355).
- Hayes-Roth, F., & McDermott, J. (1977). Knowledge Acquisition from Structural Descriptions. In *Fifth International Joint Conference on Artificial Intelligence*. (pp. 356-362).
- Hayes-Roth, F., & McDermott, J. (1978). An Interference Matching Technique for Inducing Abstractions. *Communications of the ACM*, **21**, 401-411.
- Helft, N. (1987). Inductive generalization: a logical framework. In I. Bratko & N. Lavrac (Eds.), *Progress in Machine Learning*. (pp. 149-157). Wilmslow: Sigma Press.
- Helft, N. (1988). Learning systems of first-order rules. In J. Laird (Eds.), *Proceedings of the Fifth International Conference on Machine Learning*. (pp. 395-401). Ann Arbor: Morgan Kaufmann.
- Kodratoff, Y., & Ganascia, J.-G. (1986). Improving the generalization step in learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach. Volume 2*. (pp. 215-244). Los Altos, CA: Morgan Kaufmann.
- Lavrac, N., Dzeroski, S., & Grobelnik, M. (1991). Learning Non-Recursive Definitions of Relations with LINUS. In Y. Kodratoff (Eds.), *European Working Session on Learning*. (pp. 265-281). Porto, Portugal: Springer-Verlag.
- Michalski, R. S. (1973). Discovering Classification Rules Using Variable Valued Logic System VL1. In *Third International Joint Conference on Artificial Intelligence*. (pp. 162-172).
- Michalski, R. S. (1980). Pattern Recognition as Rule-Guided Inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **2**(4), 349-361.
- Michalski, R. S. (1983). A Theory and Methodology of Inductive Learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*. Palo Alto: Tioga.
- Mozetic, I. (1987). The role of abstractions in learning qualitative models. In P. Langley (Eds.), *Proceedings of the Fourth International*

- Workshop on Machine Learning*. (pp. 242-255). Irvine, CA: Morgan Kaufmann.
- Muggleton, S. (1987). Duce, An oracle based approach to constructive induction. In *Proceedings of the International Joint Conference on Artificial Intelligence*. (pp. 287-292). Milan: Morgan Kaufmann.
- Muggleton, S., & Buntine, W. (1988). Machine invention of first-order predicates by inverting resolution. In R. S. Michalski, T. M. Mitchell, & J. G. Carbonell (Eds.), *Proceedings of the Fifth International Machine Learning Conference*. (pp. 339-352). Ann Arbor, Michigan: Morgan Kaufmann.
- Muggleton, S., & Feng, C. (1990). Efficient induction of logic programs. In *First Conference on Algorithmic Learning Theory*, Tokyo: Omsa.
- Muggleton, S. (1991). Inductive Logic Programming. *New Generation Computing*, **8**, 295-318.
- Pennypacker, J. C. (1963). *An Elementary Information Processor for Object Recognition* (SRC No. 30-I-63-1). Case Institute of Technology.
- Plotkin, G. D. (1970). A Note on Inductive Generalization. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence 5*. (pp. 153-163). Edinburgh University Press.
- Plotkin, G. D. (1971a). A further note on inductive generalization. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence 6*. New York: Elsevier.
- Plotkin, G. D. (1971b) *Automatic Methods of Inductive Inference*. Ph.D. Thesis, Edinburgh University.
- Quinlan, J. R. (1986). Induction of Decision Trees. *Machine Learning*, **1**, 81-106.
- Quinlan, J. R. (1990). Learning Logical Definitions from Relations. *Machine Learning*, **5**, 239-266.
- Reynolds, J. C. (1970). Transformational Systems and the Algebraic Structure of Atomic Formulas. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence 5*. (pp. 153-163).
- Rouveirol, C., & Puget, J.-F. (1990). Beyond Inversion of Resolution. In *Proceedings of the Seventh International Conference on Machine Learning*, Morgan Kaufmann.
- Sammut, C. A. (1981). *Learning Concepts by Performing Experiments*. Ph.D. Thesis, Department of Computer Science, University of New South Wales.

- Sammut, C. A. (1985). Concept Development for Expert System Knowledge Bases. *Australian Computer Journal*, **17**(1).
- Sammut, C. A., & Banerji, R. B. (1986). Learning Concepts by Asking Questions. In R. S. Michalski Carbonell, J.G. and Mitchell, T.M. (Eds.), *Machine Learning: An Artificial Intelligence Approach, Vol 2*. (pp. 167-192). Los Altos, California: Morgan Kaufmann.
- Shapiro, E. Y. (1981a). An Algorithm that Infers Theories from Facts. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, (pp. 446-451). Vancouver: Morgan Kaufmann.
- Shapiro, E. Y. (1981b). *Inductive Inference of Theories From Facts* (Technical Report No. 192). Yale University.
- Srinivasan, A., Muggleton, S., & Bain, M. (1992). Distinguishing Exceptions from Noise in Non-monotonic Learning. In S. Muggleton (Ed.), *Second International Workshop on Inductive Logic Programming*, Tokyo:
- Vere, S. (1975). Induction of Concepts in the Predicate Calculus. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*. (pp. 351-356).
- Vere, S. A. (1977). Induction of Relational Productions in the Presence of Background Information. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*.
- Vere, S. A. (1978). Inductive Learning of Relational Productions. In D. A. W. a. F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems*. (pp. 281-295). New York: Academic Press.
- Vere, S. A. (1980). Learning Disjunctive Concepts. Personal Communication.
- Vere, S. A. (1980). Multilevel Counterfactuals for Generalizations of Relational Concepts and Productions. *Artificial Intelligence*, **14**(2), 139-164.
- Vere, S. A. (1981). *Constrained N-to-1 Generalizations* (Technical Report) Jet Propulsion Laboratory, Pasadena.