Logic and learning: Turing's legacy

Stephen Muggleton,
Oxford University Computing Laboratory,
11 Keble Road,
Oxford, OX1 3QD,
UK.

Abstract

Turing's best known work is concerned with whether universal machines can decide the truth value of arbitrary logic formulae. However, in this paper it is shown that there is a direct evolution in Turing's ideas from his earlier investigations of computability to his later interests in machine intelligence and machine learning. Turing realised that machines which could learn would be able to avoid some of the consequences of Gödel's and his results on incompleteness and undecidability. Machines which learned could continuously add new axioms to their repertoire. Inspired by a radio talk given by Turing in 1951, Christopher Strachey went on to implement the world's first machine learning program. This particular first is usually attributed to A.L. Samuel. Strachey's program, which did rote learning in the game of Nim, preceded Samuel's checker playing program by four years. Neither Strachey's nor Samuel's system took up Turing's suggestion of learning logical formulae. Developments in this area were delayed until Gordon Plotkin's work in the early 1970's. Computer-based learning of logical formulae is the central theme of the research area of Inductive Logic Programming, which grew directly out of the earlier work of Plotkin and Shapiro. In the present paper the author describes the state of this new field and discusses areas for future development.

1 Alan Turing and the history of logic and learning

1.1 The Hilbert program

At the 1928 International Mathematical Congress David Hilbert, one of the greatest mathematicians of the previous thirty years, set out three central questions for logic and mathematics. Was mathematics

1. *complete* in the sense that every mathematical statement could either be proved or disproved,

2. *consistent* in the sense that false statements could never be derived by a sequence of valid steps and

3. *decidable* in the sense that there existed a definite method which could decide the truth or falsity of any mathematical assertion?

Hilbert expected a positive answer to all three questions. Within three years Kurt Gödel [12] had shown that not even arithmetic could be both complete and consistent. Within a decade both Alonzo Church [6] and Alan Turing [42] had shown the undecidability of certain mathematical assertions.

Turing's solution to this problem was based on defining a machine which emulated an ideal human mathematician who calculated with a pen and a one-dimensional roll of paper. A universal machine was one which could, when loaded with the appropriate definitions, simulate any other computing machine. Turing had purposefully devised a machine which had abilities equivalent to a human mathematician. However Gödel's result had been proved by showing that certain statements which were evidently true to a human could not be proved within a limited logical system. It seems that Turing noticed the clash for in his PhD. thesis on ordinal logics (published in 1939 [43]) Turing attempted to circumvent Gödel's result. The idea was to introduce a set of 'oracles', each capable of deciding the truth of unprovable statements. The use of such oracles would allow for a complete logic. However they required an element of non-mechanical *intuition*. This approach failed to reconcile Turing's belief in purely mechanical intelligence with Gödel's incompleteness result.

1.2 The war period

It would appear that an alternative solution to the incompleteness problem presented itself to Turing through his war-time work as a cryptographer. Later, in 1948 [45], he was to say that

> There is a remarkably close parallel between the problems of the physicist and those of the cryptographer. The system on which a message is enciphered corresponds to the laws of the universe, the intercepted messages to the evidence available, the keys for a day or a message to important constants which have to be determined.

Clearly, the physicist, like the cryptographer is continuously completing his theories. By learning from experience intelligent machinery should be capable of avoiding many of the problems of Gödel's incompleteness results. The area of scientific discovery alluded to by Turing in the above quote is the theme of this present Machine Intelligence workshop. The workshop paper by Sternberg et al.

[39] describes recent progress in applying machine learning techniques to discovery of new scientific knowledge.

At Bletchley Park, in 1943, in numerous out-of-hours contexts Turing discussed the problem of machine intelligence with both Donald Michie and Jack Good. According to Andrew Hodges [13], Turing's biographer,

> These meetings were an opportunity for Alan to develop the ideas for chess-playing machines that had begun in his 1941 discussion with Jack Good. They often talked about mechanisation of thought processes, bringing in the theory of probability and weight of evidence, with which Donald Michie was by now familiar. ... He (Turing) was not so much concerned with the building of machines designed to carry out this or that complicated task. He was now fascinated with the idea of a machine that could *learn*.

According to Michie, Turing put more effort into thinking about learning than anyone else in the group. He circulated for comment an unpublished typescript covering his ideas. Unfortunately it appears that this manuscript has not survived, though Michie believes it to have been a conceptual predecessor of Turing's 1948 N.P.L. report [45] (see section 1.5).

1.3 The Pilot ACE

At the end of the war Alan Turing joined in the race to design and implement the first general-purpose stored-program computing machine. In 1946, when working at the National Physical Laboratory, Turing authored the ACE report [44], describing a proposed 'large scale electronic digital computing machine'. The machine and its implications were described by Turing to an audience at the London Mathematical Society on 20th February 1947 [41]. The end of the lecture was dedicated to the problem of machine learning.

> Let us suppose we have set up a machine with certain initial instruction tables, so constructed that these tables might on occasion, if good reason arose, modify those tables. One can imagine that after the machine had been operating for some time, the instructions would have altered out of all recognition, but nevertheless still be such that one would have to admit that the machine was still doing very worthwhile calculations. Possibly it might still be getting results of the type desired when the machine was first set up but in a much more efficient manner.

In present day jargon this is known as speed-up learning. The machine does not increase its set of provable statements, but increases the efficiency of making the proofs. The talk by Numao et al. [25] in this workshop is aimed at speed-up learning. Turing goes on as follows

> In such a case one would have to admit that the progress of the machine had not been foreseen when its original instructions were put in. It would be like a pupil who had learnt much from his master, but had added much more by his own work. When this happens I feel that one is obliged to regard the machine as showing intelligence. As soon as one can provide a reasonably large memory capacity it should be possible to begin to experiment on these lines. .. One might reasonably hope to be able to make some real progress with a few million digits, especially if one confined one's investigation to some rather limited field such as the game of chess. It would probably be quite easy to find instruction tables which would enable the ACE to win against an average player. Indeed Shannon of Bell Telephone laboratories tells me he has won games playing by rule of thumb: the skill of his opponents is not stated. But I would not consider such a victory very significant. What we want is a machine that can learn from experience. The possibility of letting the machine alter its own instructions provides the mechanism for this, but this of course does not get us very far.

According to Turing learning is an indicator of intelligence. He realised that most experimentation with machine learning would have to wait until machines had a few million bytes of memory. Such a size of core-memory was not common until the 1970's. Turing also foresaw that chess would be an ideal domain for testing machine learning programs. Games, especially chess, have been extensively used for testing machine learning (see Bain and Muggleton's paper [2] in this workshop). Turing follows this passage, which is largely to do with speed-up learning, by resolving Gödel's incompleteness problem in a discussion of interactive inductive machine learning.

> It has for instance been shown that with certain logical systems there can be no machine which will distinguish provable formulae of the system from unprovable, .. Thus if a machine is made for this purpose it must in some cases fail to give an answer. On the other hand if a mathematician is confronted with such a problem he would search around and find new methods of proof, .. fair play must be given to the machine. Instead of it sometimes giving no answer we could arrange that it gives occasional wrong answers. .. if a machine is expected to be infallible, it cannot also be intelligent. There are several mathematical theorems which say almost exactly that. But these theorems say nothing about how much intelligence may be displayed if a machine makes no pretence at infallibility. .. No man adds very much to the body of knowledge, why should we expect more of a machine? Putting the same point differently, the machine must be allowed to

> have contact with human beings in order that it may adapt itself to their standards.

Interestingly, Shapiro's Model Inference System [37], developed in the early 1980's, works just like this. A Prolog program, which is a set of logical assertions, is interactively debugged by checking inferences against a human oracle. Shapiro divides all possible bugs in the program into three categories: incompleteness, incorrectness and non-termination. The correspondence with Hilbert's three questions about mathematics is striking, though this is not commented on by Shapiro.

1.4 Altering the program

Around this time Turing entered into an early discussion on the relative merits of neural-net learning and logic-based learning. The question is whether the structure of neural hardware is a prerequisite for machines to learn. In a letter dated 20th November 1946 [13] to W. Ross Ashby, Turing says

> It would be quite possible for the machine to try out variations of behaviour and accept or reject them in the manner you describe and I have been hoping to make the machine do this. This is possible because, without altering the design of the machine itself, it can, in theory at any rate, be used as a model of any other machine, by making it remember a suitable set of instructions. .. This theoretical possibility is attainable in practice, in all reasonable cases, at worst at the expense of operating slightly slower than a machine specially designed for the purpose in question. Thus, although the brain may in fact operate by changing its neuron circuits by the growth of axons and dendrites, we could nevertheless make a model, within the ACE, in which the possibility was allowed for, .., instead of building a special machine.

Turing clearly sees the necessity of making use of an interpreter in machine learning. This allows the learning program to be distinguished from the program being learned and is nowadays standard practice. The idea of using an interpreter within learning may seem obvious now but it solved a paradox going back over a hundred years. In 1842, Ada Lovelace [17], describing Babbage's planned Analytical Engine noted that

> The Analytical Engine has no pretensions whatever to *originate* anything. It can do *whatever we know how to order it* to perform.

In the following passage from his 1950 Mind article [46] Turing describes how secondary (interpreted) rules can be originated.

> How can the rules of a machine change? They should describe completely how the machine will react whatever its history might be, whatever changes it might undergo. The rules are thus quite time invariant .. The explanation of the paradox is that the rules which get changed in the learning process are of a rather less pretentious kind, claiming only an ephemeral validity. The reader may draw a parallel with the Constitution of the United States.

1.5 The 1948 NPL report

Prior to his resignation from the National Physical Laboratory in 1948, Turing submitted a report [45] devoted almost entirely to his ideas on machine learning. Michael Woodger says that Turing's report caused a furore at N.P.L with his prognostications of intelligent machinery. 'Turing is going to infest the countryside' some declared 'with a robot which will live on twigs and scrap iron'. They were reacting to the following section in Turing's report.

> In order that the machine should have a chance of finding things out for itself it should be allowed to roam the countryside, and the danger to the ordinary citizen would be serious. .. although this method is probably the 'sure' way of producing a thinking machine it seems to be altogether too slow and impractical.

Though machines learning from interacting with the real world must have sounded wild at the time, it is similar in spirit to the experiments in learning to fly a plane and balancing a pole described in this workshop by Michie and Camacho [18] and Sammut [33]. The present work grew out of investigations by Michie and Chambers [19] in the 1960's. Turing cautions about the danger to citizens and the slowness and impracticality of on-line learning from real-world data. The research reported in this workshop avoids this problem by carrying out all learning within the safe confines of a simulator.

Much of what Turing wrote in the 1948 report is still relevant to present research in machine learning. Returning once more to the connection between learning and Gödel's incompleteness theorem Turing writes as follows.

> The argument from Gödel's and other theorems rests essentially on the condition that the machine must not make mistakes .. Gauss was asked at school to do the addition 15+18+21+...+54 .. and immediately wrote down 483, presumably calculating it as (15+54)(54-12)2.3. .. imagine a situation where the children were given a number of additions to do, of which the first 5 were all arithmetic progressions, but the 6th was say 23+34+45+ ... +100+112+122+ ... 199. Gauss might have given the answer to this as if it were an arithmetic progression, not having noticed that the 9th term was 112 instead of 111.

This is a neat exposition of the problem of noise in learning. The question is whether it is better to learn a simple rule which correctly covers 5 out of the 6 progressions or to make an exception of the last incorrectly coded progression. This topic is addressed in the papers by Tsukimoto and Morita [40] and Srinivasan et al. [38] in this workshop.

In the report Turing describes the development of the human infant cortex as that of transforming an unorganised machine into a universal one. He then goes on to describe an experiment in which, using a hand-simulated program, he managed to train an unorganised state-transition machine to become a universal Turing machine. The result is a machine that *learns how to accept and interpret instructions.* This goes beyond the aspirations of almost all modern machine learning algorithms. Commenting on this experiment Turing notes that

> One particular kind of phenomenon I had been hoping to find in connection with the P-type machines. This was the incorporation of old routines into new. One might have 'taught' (i.e. modified or organise) a machine to add (say). Later one might teach it to multiply by small numbers by repeated addition and so arrange matters that the same set of situations which formed the addition routine, as originally taught, was also used in the additions involved in the multiplication.

Here Turing is discussing the problem of incremental learning with background knowledge. The discussion of learning plus and then multiply is very reminiscent of Sammut's Marvin program [34] which was taught in exactly this way.

In the following passage Turing discusses child development with relationship to machine learning.

> The training of the human child depends largely on a system of rewards and punishments, and this suggests that it ought to be possible to carry through the organising with only two interfering inputs, one for 'pleasure' or 'reward' (R) and the other for 'pain' or 'punishment' (P).

This is equivalent to the use of 'positive' and 'negative' examples in supervised machine learning. Later in his 1950 Mind article [46] Turing commented on this system of punishment and reward.

> The use of punishments and rewards can at best be a part of the teaching process. Roughly speaking, if the teacher has no other means of communicating to the pupil, the amount of information which can reach him does not exceed the total number of rewards and punishments applied. .. It is necessary therefore to have some other 'unemotional' channels of communication. If these are available it is possible to teach a machine by punishments and rewards to obey orders given in some language, e.g. symbolic language. .. The use of

> this language will diminish greatly the number of punishments and rewards required.

There is a clear understanding here of the information amplifying effect of symbolically encoded background knowledge. Without it one gets only one bit of information per example. The following passage from the 1948 N.P.L. report makes clear the kind of symbolic language that Turing intended.

> Starting with a UPCM (Universal Practical Computing Machine) we first put a program into it which corresponds to building in a logical system (like Russell's *Principia Mathematica*). This would not determine the behaviour of the machine completely ..

The language intended for learning is predicate calculus. This is the basis of what is now called 'Inductive Logic Programming' (ILP). Rather than building in Russell's Principia, researchers in this area generally make use of a Prolog theorem-proving interpreter. This provides the power of first-order predicate calculus through its use of methods based on Alan Robinson's [31] resolution theorem-proving. ILP is discussed in Section 2.

1.6 Christopher Strachey and the first machine learning program

On 15th May 1951 Turing gave a lecture entitled 'Can Digital Machines Think' on the BBC's Third Programme. According to Hodges [13]

> This short talk did not include any details of how he proposed to program a machine to think, beyond the remark that 'it should bear a close relation to that of teaching.' This comment sparked off an immediate reaction in a listener: Christopher Strachey ..

In May 1951 Strachey was working on writing a draught's program on the recently[1] working Pilot ACE machine. On the evening of the broadcast Strachey wrote to Turing.

> ... The essential thing which would have to be done first, would be to get the machine to programme itself from very simple and general input data ... It would be a great convenience to say the least if the notation chosen were intelligible as mathematics ... once the suitable notation is decided, all that would be necessary would be to type more or less ordinary mathematics and a special routine called,

[1] According to Mike Woodger [47] the first program ran on the Pilot ACE in May 1950. However, the Pilot ACE operated without component error for more than half an hour for the first time in September 1950. The first 'large' program, which solved 17 simultaneous linear equations, ran on 26th June 1951.

> say, 'Programme' would convert this into the necessary instructions to make the machine carry out the operations indicated. This may sound rather Utopian, but I think it, or something like it, should be possible, and I think it would open the way to making a simple learning programme. I have not thought very seriously about this for long, but as soon as I have finished the Draughts programme I intend to have a shot at it.

Strachey decided to test his ideas on learning with the game of Nim. In this game three piles of matches are laid out, and two players take turns to remove as many matches as they want from any pile. A non-mathematical friend of Strachey's had noticed that any player who could achieve the position (n,n,0) had won, since it was only necessary to copy the opponents moves to reduce the heaps to (0,0,0). According to Hodges [13]

> He (Strachey) had worked out a program which could keep a record of winning positions, and so improve its play by experience, but it could only store them individually, as (1,1,0), (2,2,0) and so on. This limitation soon allowed his novice friend to beat the program.

Strachey wrote

> This shows very clearly, I think, that one of the most important features of thinking is the ability to spot new relationships when presented with unfamiliar material ...

Strachey's was a simple rote learning program. It appears to the author to be the first implemented machine learning program. Strachey's comments show that he had understood the importance of generalisation within machine learning. In machine learning the the patterns of (1,1,0) and (2,2,0) can be generalised to (n,n,0) using Plotkin's [27] least general generalisation operator. Interestingly, Strachey's Nim rule is almost isomorphic to the colinearity rule in the King-Rook-King illegality domain described in [22]. Machine learning of the colinearity rule has been shown [22] to require relational learning (ILP).

Later A.L. Samuel, in the USA, was also to start work on a draughts (or checkers) playing program. According to Donald Michie, Samuel inherited Strachey's draughts playing program. Samuel published a technical report in 1955 in which he had incorporated Strachey's ideas on rote learning into the checker player. In his 1959 report [35] the rote learning approach had been extended by the use of parameter learning. By the late 1960's Samuel [36] had extended this method even further to incorporate elements of simple logic learning.

2 Inductive logic programming

In this workshop Alan Robinson gave the opening paper [30] on Turing and the history of computation. It is a great honour to have been asked to give the clos-

ing paper on Turing and machine learning. As every computer scientist knows Alan Robinson's 1965 paper [31] on machine-oriented theorem proving has had and continues to have an enormous effect on computer science and artificial intelligence. The modern subject of Logic Programming is based on Robinson's theorem-proving techniques. Prolog is the language at the centre of Logic Programming.

For the following reasons pure Prolog is also an almost ideal target language for symbolic learning.

- Prolog has the expressiveness of first-order predicate calculus. It is thus capable of expressing grammars, plans, mathematical and scientific theories as well as arbitrary computer programs.
- Prolog programs consist of conjunctions of clauses. Logical conjunction is both associative and commutative. Because of this clauses represent independent axioms which can be added to in any order.
- Pure Prolog programs, treated as sets of logical clauses, have a clear and simple semantics [16].
- Prolog can be efficiently interpreted using SLD resolution. The importance of an interpreter within a learning system was noted in Section 1.4.

Inductive Logic Programming (ILP) is a research area formed at the intersection of Machine Learning and Logic Programming. ILP systems develop predicate descriptions from examples and background knowledge. The examples, background knowledge and final descriptions are all described as logic programs. A unifying theory of Inductive Logic Programming is being built up [20] around lattice-based concepts such as refinement [37, 9], least general generalisation [28, 20], inverse resolution [23] and most specific corrections [1]. In addition to a well established tradition of learning-in-the-limit convergence results [28, 37, 7], some results within Valiant's PAC-learning framework have been demonstrated for ILP systems [26, 10].

2.1 Theory

In the general setting an ILP system S will be given a logic program B representing background knowledge and a set of positive and negative examples $\langle E^+, E^- \rangle$, typically represented as ground literals. In the case in which $B \not\models E^+$, S must construct a clausal hypothesis H such that

$$B \wedge H \models E^+$$

where B, H and E^- are satisfiable. In some approaches [37, 29] H is found via a general-to-specific search through the lattice of clauses. This lattice is rooted

at the top by the empty clause (representing falsity) and is partially ordered by θ-subsumption (H θ-subsumes H' with substitution θ whenever $H\theta \subseteq H'$). Two clauses are treated as equivalent when they both θ-subsume each other.

Following on from work by Plotkin [28], Buntine [5] demonstrated that the equivalence relation over clauses induced by θ-subsumption is generally very fine relative to the the equivalence relation induced by entailment between two alternative theories with common background knowledge. Thus when searching for the recursive clause for *member/2* (list-membership), infinitely many clauses containing the appropriate predicate and function symbols are θ-subsumed by the empty clause. Very few of these entail the appropriate examples relative to the base case for *member/2*.

Specific-to-general approaches based on Inverse Resolution [23, 32, 34] and relative least general generalisation [5, 24] maintain admissibility of the search while traversing the coarser partition induced by entailment. For instance Inverse Resolution is based on inverting the equations of Robinson's resolution operator to find candidate clauses which resolve with the background knowledge to give the examples. Inverse resolution [23] can also be used to add new theoretical terms (predicates) to the learner's vocabulary. This process is known as *predicate invention*.

2.2 ILP applications

Many of the ILP applications to date have been developed using Muggleton and Feng's [24] Golem. In [4], Bratko and Muggleton showed that simple naive physics systems could be learned within an ILP setting. In this case, a qualitative model of water-filled U-tube was learned by Golem from 5 positive examples 6 negative examples and background knowledge representing Kuiper's [15] QSIM theory.

In [11] Cao Feng used Golem to construct a complete and correct set of diagnostic rules from a qualitative model of the power subsystem of a European Space Agency satellite.

Dolsak and Muggleton [8] used Golem to construct design rules for finite element analysis used within CAD packages.

Golem has also had two major successes in discovering new scientific knowledge in the area of biomolecular modelling. Firstly [21] in the area of predicting protein secondary structure from primary amino acid sequences Golem produced reasonably intelligible rules with a higher accuracy than other approach tested on the same domain. Secondly [14] in a drug design domain Golem produced structure-activity prediction rules with an accuracy which is at least as good as the industry-standard Hansch regression technique. The advantage of the Golem approach lies in the fact that the rules are much easier for medicinal chemists to understand. These domains are discussed in detail in the paper within this workshop by Sternberg et al. [39].

3 Future trends and developments

In his writings Alan Turing did not make the modern distinction between computer science and artificial intelligence. The universal Turing machine was in fact inspired by Turing's idea of an automatic mathematician. It would therefore not have surprised him in the least to see automatic theorem proving at the heart of Logic Programming. With this achievement firmly established it seems reasonable to take seriously Turing's other aspirations.

In Section 1 we showed that Turing saw machine learning as a central component in the future of computing. Present day machine learning is a specialised subject area of Artificial Intelligence. One way to increase the impact of machine learning might be to develop specialised conceptualising tools as Scientific Assistants. Such a tool would help scientists by suggesting interesting hypotheses from data and background knowledge, as has already been started in herbicide selection, cardiology [3] and molecular chemistry [21, 14, 39]. This kind of tool certainly seems like something worth aiming for.

However in Turing's vision of learning machines, the learning played a much more fundamental role. Every action involving communication between humans and computers, and even between one computer and another has the potential for triggering learning processes. It might be possible to achieve Turing's aims by making ILP an integral part of machine interfaces. Learning should eventually become to user-interfaces what theorem proving has become to program execution within a Logic Programming framework.

Shapiro's [37] debugging system made a start in this direction. However his system was inefficient and was never incorporated into any widely-used program development system. Present ILP systems lack a standard model for their implementation. Although FOIL [29] and Golem [24] are reasonably widely used and efficient, their approaches differ considerably.

If we were to take humans and animals as our model of computation then learning should be a part of every information processing task within computers. This seems like a very tall order. However, integrating learning sufficiently strongly into Logic Programming would in a sense achieve this end. The Japanese Fifth Generation project proved that Logic Programming could be used throughout an operating system. Although today's logic programming systems are likely to be obsolete within a decade, logic will maintain and increase its role within computing. It has been fifty years since Turing's initial investigations of logic and learning. Powerful logic-based learning systems will play a vital and increasingly central part within the next fifty years.

Acknowledgements.

Thanks are due to Donald Michie for valuable information concerning Alan Turing's war-time discussions. His inspiration and support have helped generations

of young scientists to share Alan Turing's vision.

References

[1] M. Bain and S. Muggleton. Non-monotonic learning. In D. Michie, editor, *Machine Intelligence 12*. Oxford University Press, 1991.

[2] M. Bain and S. Muggleton. Learning optimal chess endgame strategies. In *Machine Intelligence 13*. (to appear).

[3] I. Bratko, I. Mozetic, and N. Lavrac. *KARDIO: a study in deep and qualitative knowledge for expert systems*. MIT Press, Cambridge, 1989.

[4] I. Bratko, S. Muggleton, and A. Varsek. Learning qualitative models of dynamic systems. In S. Muggleton, editor, *Inductive Logic Programming*, pages 437–452. Academic Press, London, 1992.

[5] W. Buntine. Generalised subsumption and its applications to induction and redundancy. *Artificial Intelligence*, 36(2):149–176, 1988.

[6] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[7] L. de Raedt and M. Bruynooghe. An overview of the interactive Concept-Learner and Theory Revisor CLINT. In S. Muggleton, editor, *Inductive Logic Programming*, pages 163–192, London, 1992. Academic Press.

[8] B. Dolsak and S. Muggleton. The application of inductive logic programming to finite element mesh design. In S. Muggleton, editor, *Inductive Logic Programming*, pages 453–472, London, 1992. Academic Press.

[9] S. Dzeroski and N. Lavrac. Refinement graphs for FOIL and LINUS. In S. Muggleton, editor, *Inductive Logic Programming*, pages 319–334. Academic Press, London, 1992.

[10] S. Dzeroski, S. Muggleton, and S. Russell. Pac-learnability of determinate logic programs. In *Proceedings of the International Conference on Learning Theory (COLT92)*, San Mateo, Calif., 1992. Kaufmann.

[11] C. Feng. Inducing temporal fault dignostic rules from a qualitative model. In S. Muggleton, editor, *Inductive Logic Programming*, pages 473–494. Academic Press, London, 1992.

[12] K. Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter System I. *Monats. Math. Phys.*, 32:173–198, 1931.

[13] A. Hodges. *The enigma of intelligence.* Unwin Paperbacks, Hemel Hempstead, 1985.

[14] R. King, S. Muggleton, R. Lewis, and M. Sternberg. Drug design by machine learning: the use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the National Academy of Sciences*, 1992. To appear.

[15] B. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289–338, 1986.

[16] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1984.

[17] A. Lovelace. Lady Lovelace's memoir on the Analytical Engine. In B.V. Bowden, editor, *Faster than thought.* 1953, London, 1953. Republished document from 1842.

[18] D. Michie and R. Camacho. Building symbolic represenatations of intuitive real-time skills from performance data. In *Machine Intelligence 13.* (to appear).

[19] D. Michie and R.A. Chambers. BOXES: An experiment in adaptive control. In E. Dale and D. Michie, editors, *Machine Intelligence 2*, pages 137–152. Oliver and Boyd, Edinburgh, 1968.

[20] S. Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[21] S. Muggleton, R. King, and M. Sternberg. Predicting protein secondary-structure using inductive logic programming, 1992. submitted to Protein Engineering.

[22] S.H. Muggleton, M.E. Bain, J. Hayes-Michie, and D. Michie. An experimental comparison of human and machine learning formalisms. In *Proceedings of the Sixth International Workshop on Machine Learning.* Kaufmann, 1989.

[23] S.H. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 339–352. Kaufmann, 1988.

[24] S.H. Muggleton and C. Feng. Efficient induction of logic programs. In S. Muggleton, editor, *Inductive Logic Programming*, pages 281–298, London, 1992. Academic Press.

[25] M. Numao, M. Takashi, and M. Shimura. Inductive speed-up learning of logic programs. In *Machine Intelligence 13.* (to appear).

[26] C. Page and A. Frisch. Generalisation and learnability: a study of constrained atoms. In S. Muggleton, editor, *Inductive Logic Programming*, pages 29–62, London, 1992. Academic Press.

[27] G. Plotkin. A note on inductive generalisation. In B. Meltzer and D. Michie, editors, *Mchine Intelligence 5*, pages 153–164. Edinburgh University Press, Edinburgh, 1969.

[28] G.D. Plotkin. *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University, August 1971.

[29] R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[30] A. Robinson. Turing, von Neumann and the universal machine. In *Machine Intelligence 13*. (to appear).

[31] J.A. Robinson. A machine-oriented logic based on the resolution principle. *JACM*, 12(1):23–41, January 1965.

[32] C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In S.H. Muggleton, editor, *Inductive Logic Programming*. Academic Press, London, 1992.

[33] C. Sammut. Recent progress with BOXES. In *Machine Intelligence 13*. (to appear).

[34] C. Sammut and R.B Banerji. Learning concepts by asking questions. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach. Vol. 2*, pages 167–192. Kaufmann, Los Altos, CA, 1986.

[35] A.L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3:211–229, 1959.

[36] A.L. Samuel. Some studies in machine learning using the game of checkers, 2 recent progress. *IBM Journal of research and development*, 11:601–617, 1967.

[37] E.Y. Shapiro. *Algorithmic program debugging*. MIT Press, 1983.

[38] A. Srinivasan, S. Muggleton, and M. Bain. The justification of logical theories. In *Machine Intelligence 13*. (to appear).

[39] M. Sternberg, R. King, and S. Muggleton. Machine learning and biomolecular modelling. In *Machine Intelligence 13*. (to appear).

[40] H. Tsukimoto and C. Morita. The discovery of propositions in noisy data. In *Machine Intelligence 13*. (to appear).

[41] A. Turing. Lecture to the London Mathematical Society on 20 February 1947. Published in 'A.M. Turing's ACE Report of 1946 and other papers', MIT Press.

[42] A. Turing. On computable numbers with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1937.

[43] A. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, pages 161–228, 1939.

[44] A. Turing. Proposal for development in the mathematics division of an Automatic Computing Engine (ACE). Technical report, National Physical Laboratory, 1946. Published in 'A.M. Turing's ACE Report of 1946 and other papers', MIT Press.

[45] A. Turing. Intelligent machinery. Technical report, National Physical Laboratory, 1948. First published in Machine Intelligence 5, Edinburgh University Press.

[46] A. Turing. Computing machinery and intelligence. *Mind*, 1950. Reprinted in 'The Mind's I' ed. D. Hofstadter and D. Dennett, pub. by Basic Books, New York, 1981.

[47] M. Woodger. The history and present use of digital computers at the National Physical Laboratory. *Process Control and Automation*, pages 437–442, November 1958. Reprinted in 'A.M. Turing's ACE Report of 1946 and other papers', MIT Press.