LEARNING LOGICAL EXCEPTIONS IN CHESS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF STATISTICS AND MODELLING SCIENCE

AND THE COMMITTEE FOR POSTGRADUATE STUDIES

OF THE UNIVERSITY OF STRATHCLYDE, GLASGOW,

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Michael Bain

1994

Abstract

This thesis is about inductive learning, or learning from examples. The goal has been to investigate ways of improving learning algorithms. The chess end-game "King and Rook against King" (KRK) was chosen, and a number of benchmark learning tasks were defined within this domain, sufficient to over-challenge state-of-the-art learning algorithms. The tasks comprised learning rules to distinguish (1) illegal positions and (2) legal positions won optimally in a fixed number of moves.

From our experimental results with task (1) the best-performing algorithm was selected and a number of improvements were made. The principal extension to this generalisation method was to alter its representation from classical logic to a non-monotonic formalism. A novel algorithm was developed in this framework to implement rule specialisation, relying on the invention of new predicates. When experimentally tested this combined approach did not at first deliver the expected performance gains due to restrictions on the size of training samples. With larger samples and a more efficient generalisation method, together with the new specialisation algorithm, a successful result was obtained.

The outcome of this test was a new inductively generated solution to the KRK illegality problem. Unlike other solutions using similar formulations of this problem in the Machine Learning literature the new result was complete and correct. The key step was the invention of new predicates during the specialisation stage of the inductive process.

This new method was then applied to task (2), i.e. learning rules to conjecture the value of n for positions won optimally in n moves. Positions were extracted from an exhaustive database containing all canonical positions in the KRK end-game. Prolog rules induced from these are presented for positions won in 0 and 1. These are complete and correct when tested against the database, and have been validated by a human expert.

Acknowledgements

I owe a debt of gratitude to many who have helped me along the way to completing this thesis. Without them, it would not exist. My thanks are specifically due as follows.

To Donald Michie, who first employed me at the Turing Institute to work on the US Army chess project. Beyond giving me my first job in Artificial Intelligence, by his consistent interest in and many excellent suggestions concerning this work his involvement has proved invaluable. I will always feel privileged to have worked with him.

To Stephen Muggleton, who has been both colleague and friend. His inspiration, technical brilliance and communicable insight are by now well known throughout the Machine Learning community, as they have been to me for many years. He proposed the chess problems studied in the thesis, and suggested ways in which they might be approached. It is largely due to his encouragement, foresight and patience that I have stayed the course. Together with Donald Michie, he has taught me that AI is of relevance and importance to the heart of the scientific enterprise.

To Bob Henery, for very generously agreeing to undertake the supervision of this thesis.

To Jean Hayes-Michie, for her collaboration on the human learning experiments, and her empathy as a fellow PhD candidate.

To Thirza Castello-Cortes, for her kindness and forbearance throughout many hours of detailed supervision.

To Ivan Bratko, for his constructive criticism at all stages of this work and in particular his help in validating the induced chess concepts.

To Arthur van Hoff for his work on the KRK database.

To Ashwin Srinivasan for many fruitful conversations and his moderating influence on my ideas.

I would like to thank the staff, especially Mary Ritchie, and the pupils of Glenwood School, Glasgow, for their help in carrying out the human learning experiments.

Thanks also to David Aha, Robin Boswell, Wray Buntine, Jason Catlett, Peter Cheeseman, Peter Clark, Saso Dzeroski, Cao Feng, Boonserm Kijisikurul, Ross King, Eduardo Morales, Tim Niblett, Ross Quinlan, Stuart Russell, Claude Sammut, Barry Shepherd, Sia Tangkitvanich, Stefan Wrobel and any others not mentioned above with whom I have had discussions regarding the material in this thesis.

I am especially grateful to all the staff of the Turing Institute for providing the facilities allowing the conduct and completion of this research. I would also like to thank Knowledgelink and Attar Software for facilities and help. All drawings were produced using Tgif Version 2.9 by William Chia-Wei Cheng.

This work was supported in part by Alvey project IKBS/092 and IED project 4/1/1320.

Finally I thank those of my family and friends who, with varying degrees of wit or seriousness, expressed such scepticism as to the eventual completion of this thesis that my determination to finish the work was thereby increased.

Previous Publication

Parts of Chapter 4 appeared in [85]. Parts of Chapter 5 and Appendix B appeared in [5]. A version of Chapter 6 appeared as [3]. Parts of Chapter 7 appeared in [4].

Authorship

The work in this thesis is my own. The composition of the thesis is my own. However, with respect to the inclusion of material published previously in collaboration with different authors, I acknowledge the following contributions. Jean Hayes-Michie advised me on the format of the human learning experiment in Chapter 4 and Donald Michie assisted in calculating the statistics. Stephen Muggleton assisted me in proving Theorem 5.7 in Appendix B and ran Golem on the even-numbered KRK depth of win examples in Chapter 7.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Presented in this thesis are new theoretical and experimental results from work on inductive learning which involves a range of learning algorithms and includes a comparison with human learning. The work addresses the following question: could a machine learn to play chess given only example positions and some simple facts about the geometry of the board ?

Machine Learning (ML) is the science of computer programs which learn, where learning is understood as the process by which new knowledge is acquired [1]. In a number of recent commercial applications ML has delivered significant cost benefits [69, 71]. However, ML algorithms of today possess a number of capabilities. In this work some of these were empirically and theoretically characterised, with a view to constructing a systematically enhanced learning program by improving them. Using benchmark problems from a chess endgame domain, experimental trials were then repeated with this system and the results analysed with regard to implications for automation of knowledge acquisition and data analysis.

The process under study is inductive learning, or learning from examples. We interpret inductive learning as having two outcomes [72], namely:

1. improved performance (measured as "predictive accuracy") in classifying

[1] Defined in more detail Section 1.3.

unseen test data as a result of experience of training data;

2. ability to communicate the newly-acquired classification rules in human-intelligible form, i.e. "transparency".

Where a problem is complex it may not be feasible for a learner to "induce" a complete and final rule-set in a single leap, so to speak, from the level of primitive data [2]. Intermediate descriptions are commonly found in the analyses made by experienced human problem-solvers, and the creation of such descriptions can itself be seen as an inductive step on an ascent which may lead through a number of levels to the final classificatory concept [123]. This definition of the inductive process is illustrated in the diagram of Figure 1, where *both* arrows stand for inductive steps. The depiction of induction as a two-stage unidirectional process is clearly an over-simplification. For example, it ignores the iteration between hypothesis formation and experimental testing typical of theory construction, the role of analogy and the use of background knowledge. Total automation of the inductive concept formation process is a long-term goal to which this thesis is intended to contribute.

1.1 "Ant Attack"

A simple example of induction which will nonetheless be seen in Chapter 2 to be non-trivial is provided by the "Ant Attack" characterisation of one of the benchmark problems used throughout the experiments in this thesis. This induction task was designed to be presented in the form of a game, as follows. Messages are received from an army of ants by a learning agent. Each message is labelled either 'yes', which means that the ants are about to attack, or 'no' which means they are not. These messages may be encoded as simple diagrams, as in Figure 2, or as rows of letters, as in Figure 3.

[2] The terms "primitive data" and "intermediate descriptions" are discussed in more detail in Chapter 3.

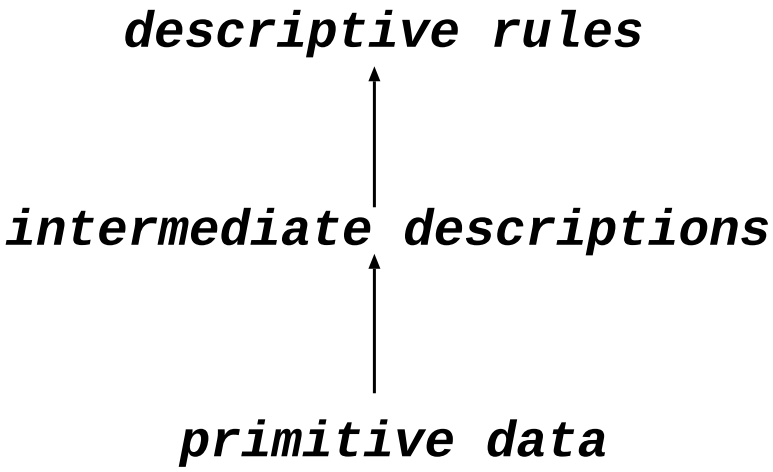


Figure 1: Levels in the inductive process.

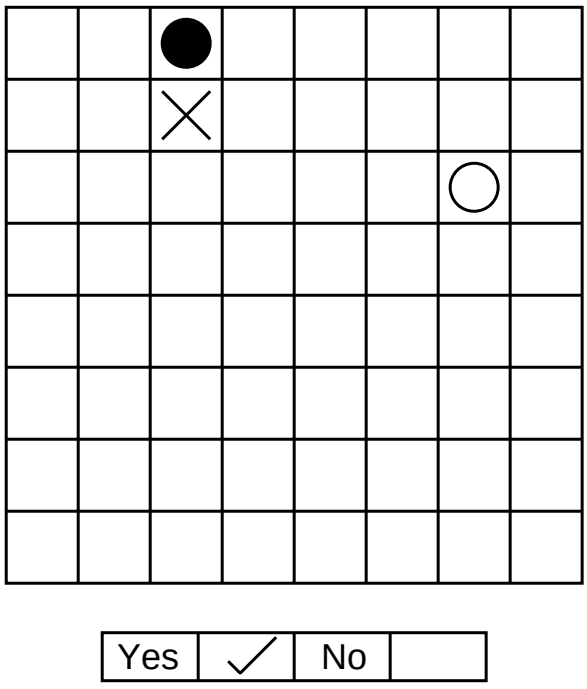
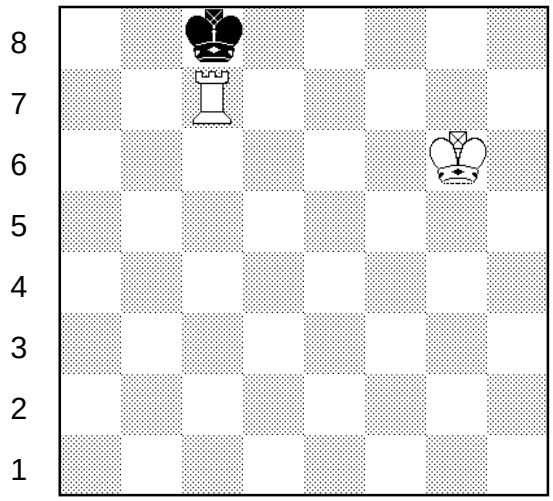


Figure 2: "Grids" representation of an 'Ant Attack' message.

						Yes	No
t	r	h	u	h	x	✓	

Figure 3: "Letters" representation of an 'Ant Attack' message.



illegal(g,6,c,7,c,8).

Figure 4: An illegal white-to-move KRK position in Prolog.

The goal of the game is to find rules which predict an "Ant Attack" from the messages. The "Ant Attack" game was invented to conceal from human subjects the fact that the actual task was learning a subset of the rules of chess from examples. Specifically, the real problem was learning to classify illegal and legal white-to-move positions in the KRK chess endgame. The messages (or examples) in fact describe chess positions using the row and column values of pieces on the board. Figure 4 shows a position and its Prolog [21] representation as supplied to one of the Machine Learning agents tested in our experiments.

Although at first glance trivial, the problem of learning rules to classify

randomly-generated KRK positions turned out to elicit enormously varied performance between different learning agents. The particular combination of capabilities in each learner can result in less than perfect performance. This brings us to the enumeration of such capabilities in the learning algorithms tested, which were broadly representative of the state-of-the-art. The following sections set out the main capabilities, relating them in some cases to particular features of the test problem chosen.

1.2 Categories of capability

We identify the following independently necessary capabilities in the learning algorithms tested.

1. Hypothesis language.
2. Theory-guided sampling.
3. Rule refinement.
4. Invention of new predicates (intermediate descriptions).
5. Noise handling.
6. Background knowledge.

1. Hypothesis language

The first capability concerns the language in which the induced rules must be expressed. This is often referred to as the "hypothesis language", defined in Appendix A. The limitations imposed by a particular hypothesis language on a learning algorithm are a major component of what is known as its "inductive bias" [34].

The hypothesis languages used by the Machine Learning algorithms tested in this thesis fall into two main categories, propositional (0^{th}-order) or relational

(1^{st}-order). These categories are based on the order of formulae which can be expressed in the language. A propositional hypothesis language is restricted to the expression of formulae in the propositional calculus, whereas a relational hypothesis language also permits first-order predicate calculus formulae. Definitions from logic are given in the Appendix in Section A.1.

Algorithms using a propositional hypothesis language suffer from two limitations:

1. the induced rules are not transparent when the concept to be learned exceeds a rather low level of complexity;

2. these algorithms typically required more examples compared with those using a relational hypothesis language to induce rules of similar accuracy.

In terms of formalism first-order logic is richer than propositional logic, with the addition of terms and predicates (see Appendix A.1). However, in the context of machine learning the issue of propositional versus first-order logic representations is not simply that of expressive power. The distinction between requiring a first-order hypothesis language over a propositional hypothesis language is dependent on the learning task.

It may be that the available domain knowledge contains atomic formulae which are adequate for the expression in propositional logic of interesting hypotheses. This was demonstrated by A. Shapiro in the case of learning a rule to classify positions in the KPa7KR chess endgame as won or not won for white [123]. In this work propositions such as "one or more black pieces control the queening square" were devised especially for the learning task by a domain expert. In Shapiro's rule this proposition is an intermediate description in the sense of Figure 1, since its truth value is computed from the positioning of individual pieces on the board (the primitive data).

However, if the learning task required automatic formation of this intermediate description in logic, expression of the finer details such as "control of the queening square" would seem to necessitate using a predicate, e.g. "piece

x attacks square y". Therefore in this context a first-order logic representation would be required.

2. Theory-guided sampling

The second capability is that of theory-guided sampling of training data. For example, in the KRK endgame the set of illegal positions can be divided into a number of intersecting subsets. Each subset contains positions which are illegal due to certain properties (sub-concepts) of the target theory. Some of these subsets are larger than others, although none completely contains another. The largest subset consists of positions where the black king is in check to the white rook. Therefore this type of position occurs more frequently in any random sample than those illegal due to, say, the adjacency of the two kings. Nearly all of the algorithms in the present work were trained on sets of randomly selected examples. Consequently, all these algorithms suffered from the fact that a very large number of training examples would be required for learning of a complete and correct definition (as defined in Chapter 5). This defect can be countered by incorporating a facility such as "windowing", first demonstrated by Quinlan [106], or the method described in Chapter 6, both of which implement a form of theory-guided sampling.

3. Rule refinement

The third capability concerns the ability to refine a provisional rule during the learning process by correcting it with respect to exceptions found in the data. With an algorithm which learns by monotonically generalising from a sequence of training examples presented singly (CIGOL [86]), ceiling effects in learning accuracy were noted in our experiments. This was due to over-generalisation by the algorithm with an accompanying inability to specialise following the occurrence of counter-examples. This defect may be remedied by the use of the non-monotonic specialisation method implemented by the CWS algorithm

developed in Chapter 5. Hypotheses formed on the basis of examples seen before a certain point in time may then be changed in the light of examples seen subsequently.

4. Invention of new predicates

The fourth capability is to change the representation during the learning process by inventing new predicates and adding them to the vocabulary of the hypothesis language. This process is the step of forming the intermediate descriptions shown in Figure 1. Two algorithms tested in the experiments (Duce [80] and CIGOL) have this ability. Unfortunately, in the training data selected by random sampling the frequency of examples which would lead these algorithms to invent new predicates was so low that no new predicates were invented. However, the invention of new predicates is also central to the CWS algorithm which was applied in the experiments to specialise hypotheses with respect to exceptions. Using a theory-guided method of sampling data to increase the relative frequency of exceptions, new predicates were invented and incorporated into the hypotheses.

5. Noise handling

There is a fifth capability, the ability to handle noise, which must be mentioned here although the experimental domains used in this study were noise-free. Three of the algorithms tested (Assistant [16], C4 [110], and CN2 [19]) were equipped with methods of handling noise in training data. An application of noise-proofing to the methods developed in Chapter 5 is reported in [79].

6. Background knowledge

The sixth capability relates to utilisation of background knowledge in the learning process. Those algorithms with input restricted to training instances described by a fixed set of attributes (Assistant, C4 and CN2) only handle background knowledge supplied in the form of extra attributes formed by some computation from the raw data. CIGOL and DUCE did not significantly incorporate background predicates in hypotheses, mainly due to search constraints. However, in the experiment with the GOLEM algorithm [87], very high performance was achieved due in part to the efficient use of background knowledge.

Before proceeding to discuss some of the relevant previous research we summarise the results from the main experiments reported in the thesis. In Chapter 4, Experiments 1a and 1b, the most effective categories of learner were found to be first CIGOL and second humans. These two learners required relatively few training examples for reasonable performance, in terms both of transparency and of accuracy. Humans clearly showed an ability to learn on the experimental task, but in general they required background knowledge beyond that available to the machine algorithms.

The results of these experiments led us to devise a new incremental learning method, based on CIGOL, incorporating non-monotonic specialisation and using theory-guided sampling of data. Experiment 1 in Chapter 6 was designed to test whether the addition of these features to overcome previously identified deficiencies would result in a level of Machine Learning superior to the human level. Instead, predictive accuracy recorded over all iterations of this incremental method first rose slightly then actually fell *below* the initial level.

This drop in performance was caused by over-generalisations of instances of illegality due to king adjacency. In Experiment 2 of Chapter 6 sufficient background knowledge was supplied to the GOLEM algorithm to enable a complete

definition of this sub-concept to be induced. The combination in this experiment of efficient first-order learning (GOLEM) and non-monotonic specialisation (CWS) resulted in a logic program of 58 clauses which correctly classified all 262,144 positions in the domain. This program is a complete and correct classification rule for KRK illegality. Note that whereas other follow-up experimentation has been carried out in other laboratories this is the first time that a 100% correct solution has been reported.

Finally in Chapter 7 the methods developed were successfully applied to the task of learning to win optimally in a subset of the KRK domain.

1.3 Historical background

In a discussion entitled 'Why should machines learn ?', Simon [128] gave the following definition:

> "Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time."

Michie [70] provided a very similar definition to this as a *weak criterion* of Machine Learning. He pointed out that the weak criterion is purely performance-based. Consequently it includes various optimisation techniques, such as statistical pattern recognition, which lie outwith the Artificial Intelligence (AI) approach to ML [15]. Motivated by the concern with knowledge and its representation within the AI community, together with the requirement that useful knowledge must be "brain-compatible" [3], he arrived at the following criterion of ML, termed *ultra-strong*:

> "The system satisfies the weak criterion and can communicate its internal updates in explicit and operationally effective symbolic form."

[3] This is taken to mean that 'a person can understand the representation and can also check it by mental application to any given example case' [123].

It is a theme of research on knowledge representation that explicit symbolic formalisms are necessary and Michie incorporated this requirement, stated as brain compatibility, in his strong criterion of ML. The criterion becomes ultra-strong when the requirement is added that knowledge be represented in operationally effective form, by which is meant that it may be used to tutor a domain expert [67]. The key practical reason for demanding that what the machine learns should be communicable to users in an explicit symbolic form lies in the fact that these users must be able to understand the decision rules in order to trust their output [74].

Adoption of this criterion of ML excludes many widely-investigated techniques from consideration. For example, under this restriction approaches within the connectionist or neural-network paradigms are excluded since they do not usually satisfy the explainability requirement, even though some work has been done on the interpretation of learned networks in symbolic languages, e.g. [26, 46]. Similarly, statistical techniques such as regression are excluded. These techniques can however perform very well with respect to the weak criterion of learning.

One of the principal strands within research on knowledge representation involves the use of logic. This usually means some variant of first-order predicate calculus [4], which we abbreviate as FOL (for "First-Order Logic"). Those who investigate issues within knowledge representation from the point of view that some form of logic is necessary have been described as "logicists". There is a well-known and continuing conflict in knowledge representation between logicists and others [60]. However, argumentation along these lines is outside the scope of our discussion, and we note only that the subset of FOL known as Horn clause logic has proved highly successful within Computer Science as the basis of the discipline of Logic Programming [135, 56].

[4] Definitions for the first-order predicate calculus are given in Appendix A, together with definitions for the propositional calculus. For convenience, the latter will occasionally be abbreviated to ZOL (for "Zeroth-Order Logic").

For the purposes of Machine Learning, Muggleton and Buntine [86] in inverting the fundamental inference procedure of machine-oriented deduction (resolution [115]) to achieve a method of induction, relied on logic as a "single, uniform knowledge representation" which "allows existing clauses to be used as background knowledge in the construction of new predicates." Clearly, therefore, in addition to strengthening links between ML and the strict logicist position on knowledge representation, they also established at a fundamental theoretical level connections to Logic Programming which have led recently to the new discipline of Inductive Logic Programming [82]. This allows work towards methods of Machine Learning which meet the above ultra-strong criterion within a logic language framework. Logic in this way provides for Machine Learning a basis for languages with well-founded syntax and semantics which are machine-executable and, up to cognitive processing limits, brain-compatible [5].

It is important to note that not all Machine Learning algorithms covered by our adopted criterion use logic as a representation language, or can as methods be best explained by mapping algorithm, input and output into a logical framework. Methods employing a decision-tree representation of the learned knowledge (hereafter referred to as tree methods) in particular have been shown to give high classificational accuracy with excellent user-transparency [107], but these have been most extensively formalised within a statistical framework (refer to [11] and cf. [72]; also [45]). For this reason, we will postpone further discussion of the logical basis of the Machine Learning algorithms developed in the thesis and used in our experiments until Chapter 5.

Despite the successes of tree methods, they have an important defect caused by reliance on a propositional hypothesis language (as discussed in Section 1.2). This is brought out clearly in the first two experiments reported in this thesis. The inadequacy of propositional languages for representing relational concepts within the domain of KRK illegality results in large numbers of examples being

[5] In the sense that "good" Prolog programs can be understood and debugged by "experienced" programmers, as can logic formulae by logicians.

required for the generation of opaque rules. How then have these methods been as successful as we have suggested ? The answer lies in the choice and definition by a sophisticated user of the domain *attributes*.

Hunt and Hovland [38] presented an algorithm to simulate human concept learning (subsequently developed by Quinlan into ID3 [107]) which modelled the learning process as reception of stimuli which were then used to formulate descriptions of the target concept. Higher-level concepts could then be described in terms of these concepts, and so on thereby ascending a hierarchy of descriptions in a scheme similar to that of Figure 1. The stimuli of their method correspond to the attributes of ID3 and its derivatives.

Quinlan [106] in the landmark paper on ID3 reported on experiments in the induction of decision-trees to classify positions in the KRKN chess endgame as "lost N-ply" (see Chapter 2). He defined a set of attributes as *inadequate* for a classification task if there are "two objects belonging to different classes but having identical values for each attribute". He concluded that "almost all the effort (for a non chess-player, at least) must be devoted to finding attributes that are adequate for the classification problem being tackled". In the same paper some proposals for automatic development of attributes were given, based on domain-specific methods of generalising the pattern of pieces on the board in selected positions. These appear not to have been successful due to the inappropriateness of the representation.

Dechter and Michie [23], motivated to widen the scope of applicability of tree methods, applied a descendant of ID3 to the induction of plans and list-processing procedures from examples. They noted the critical importance of choosing the "correct" attributes to allow the learning of these concepts. In discussing the form of input to be manipulated by the induction technique they concluded:

> "The whole potential of this approach is crucially dependent on the ability to submit to the induction mechanism the appropriate representation of the problem at hand. Briefly put, the expertise and all the domain knowledge is summarised via this set of attributes."

In addition, they reported that the restriction to a propositional representation language severely limited the utility of applying the induced rules in their respective experimental domains.

Bundy et al. [12], in a review of then currently state-of-the-art Machine Learning algorithms, which included ID3, concluded

> "All the techniques described in this paper are dependent for their success on the user-supplied description space ... Automatic provision or modification of the description space is the most urgent problem facing automatic learning."

From the three preceding references a common theme emerges. The problem with tree methods is that the user must put most of the effort expended on an induction task into developing the set of attributes. This can be disproportionate to time taken to do the induction. Quinlan [106] reported durations of three weeks and two man-months, respectively, to define the attributes for his "lost 2-ply" and "lost 3-ply" experiments. In contrast, the particular implementation of ID3 used in that work induced the decision trees in 3 and 34 seconds, respectively.

Although Quinlan's experiments with ID3 produced exact classifiers for the "lost 2-ply" and "lost 3-ply" problems, the resulting decision trees were too large to be comprehensible to domain experts. In a set of experiments on the induction of decision-tree classifiers for the KPK and KPa7KR chess endgames, A. Shapiro [123] employed the technique of "structured induction" to overcome problems of excessive tree complexity typically encountered in Quinlan's work. The effect of using the expert-supplied structure to partition the problem space prior to carrying out induction resulted in compact, though correct, trees. However, it is instructive to view the structuring process as one of top-down attribute

design. As Shapiro described it:

> "The domain expert decides on the attributes he knows to have most bearing on the problem to be solved. He takes each of these in turn and, in consultation with the programmer, they decide if these attributes are immediately codable. If any are not then the decomposition process is repeated for each attribute that is not immediately codable until none is left. This produces a hierarchical tree of subproblems whose leaf nodes are directly codable attributes."

Shapiro reported that this process was completed within 4 man-weeks for the KPa7KR problem, although the final solution required a further 6 man-weeks of refinement and correction, giving a total of 10 man-weeks.

Encoding of the attributes at leaf nodes required further work by Shapiro and the US National (FIDE) Chess Master D. Kopec. This has some relevance to our results on KRK Illegality. In building the KPa7KR classifier the attributes corresponded to humanly-perceived patterns which were difficult to compute. Accordingly attribute evaluation was based on a CLIP-emulator, a software simulation of a 2D array processor, consisting of one hundred 8×8 bit planes. As employed, this simulated special-purpose hardware for chess, with ten primitives for initialisation and propagation of bit values for piece moves (setbit, shift, etc.) and eight logical operators (and, exor, etc.) for combination of bit planes. Of course all of this had to be implemented preceding any induction of the KPa7KR classifier, and the point is relevant to the work in this thesis since the explicitly relational concepts demanded by the domain of chess were encoded in these attributes to permit induction to be carried out within the propositional formalism of ID3.

To illustrate this point a WTM position from the KPa7KR endgame is shown in Figure 5. Now for this position the boolean attribute "rimmx" (can the Black Rook be captured safely ?) happens to be false since if the White King were to take the Black Rook then the Black King would immediately capture

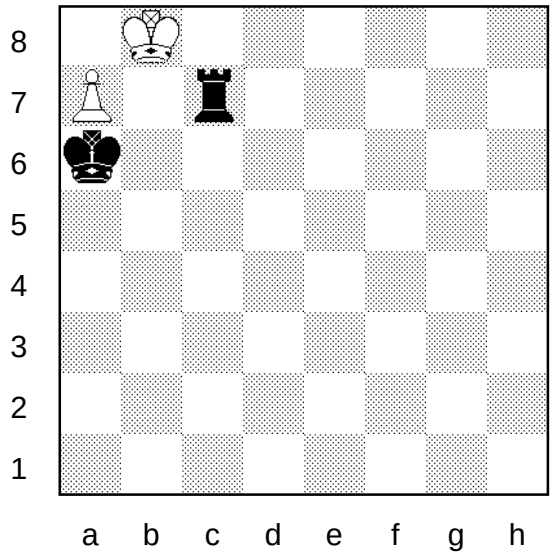


Figure 5: The relational attribute "rimmx" in KPa7KR.
This attribute encodes the feature "can the BR be captured safely ?" in the Kopec and Shapiro solution. The evaluation procedure consisted of 15 CLIP/C statements.

the White Pawn. Any position for which "rimmx" were true would be won-for-white. Although Shapiro did not publish the C-code for "rimmx", he noted that the encoding required 15 statements, consisting of calls to CLIP/C primitives or return values for explanation generation.

Once the expert-suggested structure for KPa7KR had been fully debugged using a complete database for the endgame (209718 legal positions), Shapiro used an information-theoretic method to calculate that this structure accounted for an estimated 84% of the classificatory power of the solution. Obviously this result provided a strong motivation to develop any means of automating this structuring method. One such attempt was made by Paterson [96] in the KPK chess endgame domain using a technique known as "conceptual clustering" to construct intermediate-level concepts bottom-up from lower-level examples. The work was not very successful with little significant structure being generated.

In contrast an application by Muggleton of his DUCE algorithm [80] to the problem of recreating the KPa7KR structure resulted in the machine invention of concepts similar to those in the Kopec/Shapiro solution. These were validated by independent chess experts. Muggleton reported that reconstruction from the 3196 separate example vectors supplied by Shapiro was achieved within one working day. It was this result which initiated the investigation into the application of DUCE to structure low-level position data into useful attribute-level intermediate concepts within the KRK Illegal domain which is discussed in Chapter 3.

Computer induction within a framework of formal logic was significantly advanced by the work of Plotkin [99, 100, 101] and others at the end of the 1960's and the beginning of the 1970's. At that time the impulse was to extend resolution theorem-proving technology with methods of generalisation by which new axioms could be discovered. Popplestone [103] suggested the importance of minimal (or least) generalisation of literals in induction, and that some dual of unification might be an appropriate mechanism. Reynolds [112] presented an algorithm for the "anti-unification" (the inverse of the mechanism at the heart

of resolution) of atomic formulae as a method for generalisation. Plotkin's work included an independently and simultaneously invented anti-unification method for atomic formulae and culminated in a study of the properties of the least general generalisation of clauses relative to a background theory (RLGG). The type of generalisation being discussed is illustrated in an example due to Plotkin, shown in Figure 6.

In English:

The result of heating this bit of iron to 419°C was that it melted.
The result of heating that bit of iron to 419°C was that it melted.

The result of heating any bit of iron to 419°C is that it melts.

In logic:

$\text{bit\_of\_iron}(\text{bit1}) \wedge \text{heated}(\text{bit1}, 419) \rightarrow \text{melted}(\text{bit1})$
$\text{bit\_of\_iron}(\text{bit2}) \wedge \text{heated}(\text{bit2}, 419) \rightarrow \text{melted}(\text{bit2})$

$\forall X\ \text{bit\_of\_iron}(X) \wedge \text{heated}(X, 419) \rightarrow \text{melted}(X)$

Figure 6: A typical generalisation.

The key point in this example of an intuitively appealing generalisation is the use of a standard logic. In Plotkin's approach the logical language used was similar to that of Robinson [115]. Unfortunately one of Plotkin's results was that computing RLGG could lead to the formation of a clause containing an infinite number of literals. In addition, only Least-General Generalisation (LGG) for clauses was implemented; this was rather inefficient and not tested on any large-scale tasks.

Just as the considerable research effort on theorem-proving in the late 1960's and early 1970's was accompanied by work on formal foundations of induction, so the subsequent decline in interest in general purpose theorem-provers was mirrored by a falling-off of new work on computer induction in FOL

following Plotkin's thesis. It is interesting to speculate that it was the arrival of the new discipline of Logic Programming, together with efficient methods of programming with Horn clauses in Prolog which enabled a revival of interest in general-purpose methods of induction in logic. In any case, the next work of similar stature was the thesis of E. Shapiro [126] on the induction of Horn clause theories (Prolog programs).

Shapiro's Model Inference System (MIS) had its roots, as did Plotkin's work, in ideas from the philosophy of science on theory formation. To this end, Shapiro devised an incremental framework for the generalisation and specialisation of logic programs. He then showed how this could be applied with the addition of routines for diagnosis, etc., to program synthesis and debugging. Generalisation in MIS involves searching a graph of refinements of a clause, breadth-first, from general to specific. This work has been very influential in Machine Learning, and we will discuss some aspects of it further in later chapters, but the methods were too inefficient to have found significant applications. The approach taken by Plotkin and Shapiro may usefully be classified as semantic or model-based generalisation in FOL.

In such a brief review we must necessarily omit discussion of many Machine Learning techniques which have used logical representations. However the majority of these neglected the development of a formal theoretical framework for their methods. For instance, Michalski's methods of learning in logic took the approach of constructing of a number of generalisation operators in a variant of FOL [63]. Unfortunately, no formal semantics of these methods were presented. Michalski's operators were subsequently recognised as all being variants of dropping literals from clauses, replacing constants by variables or transforming clauses with respect to background knowledge [35].

Sammut and Banerji [119] in their MARVIN system introduced a generalisation operator which transformed a set of facts in short-term memory (the example) using clauses in long-term memory (the background knowledge). This

mechanism was later shown by Muggleton [83] to be a partial method of inverting Robinson's resolution rule of deductive inference. Together with Buntine, Muggleton implemented an algorithm for "Inverse Resolution" (IR) in a Prolog system named CIGOL [86]. Subsequently, other implementations of IR have been reported by a number of workers (e.g. [116, 139]). It is useful to think of IR as a syntactic or proof-based approach to the problem of generalisation within FOL, whereas RLGG is a semantic or model-based approach.

The CIGOL system was used experimentally in a demonstration which highlighted the need for relational learning [85], some of the details of which appear in Chapter 4. In our experiment, CIGOL was efficient enough to achieve performance which outstripped competing ZOL learning algorithms on small example sets. Generally, however, for first-order target theories the IR method as implemented in CIGOL was severely hampered by the overheads of search.

An advantage of learning algorithms which employ a Horn clause logic hypothesis language is that the computational models for logic programming are well understood. Consequently, when investigating methods for the specialisation of induced CIGOL theories, the author together with S. Muggleton devised a method [5] which relies on negation as failure. This approach is called "Closed-World Specialisation", an extended treatment of which is the subject of Chapter 5. There are similarities with Vere's work on specialisation in a subset of FOL [137] using "counterfactuals". However the Closed-World Specialisation technique relies on the standard Prolog form of negation, whereas the counterfactuals method uses classical negation.

Motivated to investigate efficient induction of relational classifiers Quinlan developed an algorithm named FOIL [109]. The search strategy of FOIL is similar to Shapiro's MIS, i.e. top-down in a lattice of refinements of a clause from general to specific. In contrast to MIS, FOIL is encoded in C rather than Prolog and prunes search on the basis of an information heuristic similar to those used in ID3. FOIL was tested on the KRK Illegal problem and achieved very high performance although this was dependent on large example sets and

carefully chosen background predicates.

Quinlan also noted that the greedy heuristic search employed in FOIL can be short-sighted, and consequently cannot learn certain useful predicates. This problem is avoided in a new method of computing Plotkin's RLGG implemented in a system called GOLEM by Muggleton and Feng [87], which does not employ search. We have applied GOLEM to the KRK Illegal problem, using the same data as Quinlan's FOIL, and achieved similar high levels of performance, on which we report in Chapter 6.

In a recent paper Muggleton [82] has presented a unified framework for IR and RLGG. The framework links the methods of "inverse resolution" and "anti-unification". The significance of this result is that the syntactic and semantic approaches to induction in FOL are thereby situated within the same over-arching theory.

1.4 Thesis overview

To reiterate, the aim of the work presented in this thesis concerned extending the scope, theory and potential applicability of Machine Learning. We identified a hard problem for state-of-the-art ML algorithms, namely the induction of human-level concepts in chess from low-level descriptions of example positions. Using this domain as a source of benchmark problems we tested a range of algorithms under experimental conditions. The results of these tests led to the characterisation of a number of independently necessary capabilities for the algorithms investigated. Most of these capabilities were implemented in a new algorithm called CW-GOLEM tested in a final study on the first benchmark problem. This led to a new complete and correct inductively generated solution to the problem. Another new algorithm called GCWS, closed based on CW-GOLEM, was then tested on a second benchmark problem from the experimental domain. This resulted in success on the initial sub-problems of this more demanding benchmark test.

In Chapter 2 more relevant background is given on technical aspects of chess endgames as laboratory problems for computer induction, including generation of databases and classification of positions.

Chapter 3 presents a discussion of the KRK illegality problem and the results of the first experiment to be reported in this thesis, where the DUCE algorithm was applied to the attributes formation task.

The second experiment, a comparative study of the effect of formalism on the performance of several learning agents on the KRK illegality problem, appears in Chapter 4.

In Chapter 5 we develop a specialisation algorithm (CWS) which employs a non-monotonic representation.

This algorithm, implemented as an extension of the CIGOL system called NM-CIGOL, is experimentally evaluated on KRK illegality in the third experiment, forming Chapter 6. Chapter 6 also reports on the final experiment on KRK illegality, in which a complete solution for the problem was learned using the GOLEM algorithm in conjunction with non-monotonic specialisation in a system called CW-GOLEM.

The previous algorithm was extended and renamed GCWS. It was then applied to the task of learning rules for the target concept "black-to-move KRK position won for white in N moves". Complete and correct solutions are given for the cases where $N = 0$ and $N = 1$. This work is reported in Chapter 7.

The thesis concludes in Chapter 8 with a review of results, a summary of the relation of these results to others in the literature and a discussion of future directions following from the work presented.

Definitions of notation used in the text from logic and Machine Learning are given in Appendix A. Proofs and a note on the function "deriv" are in Appendix B. A breakdown of results obtained for induced classifiers on the sub-problems of the "black-to-move KRK position won for white in N moves" benchmark problem is in Appendix C.

Chapter 2

The KRK Chess Endgame

This chapter considers aspects of chess endgame databases and their use in experiments on computer induction. The motivation for using chess as an experimental domain is addressed. We continue by reviewing some previous work on induction in chess endgame domains. Then some basic definitions are given, and the standard algorithm for database generation is presented. The chapter concludes with a description of the KRK endgame database used throughout the remainder of this thesis.

2.1 Motivation

Why chess ? Chess is a finite game, since the rules determine that a game will eventually terminate. It is also complex, with perhaps 10^{120} continuations or paths in the complete game tree. For our purposes, however, there are further justifications. As Newell *et al.* [89] put it:

> "Let us simply assume that it is good to know how to do mechanically what man can do naturally — both to add to man's knowledge of man, and to add to his kit of tools for controlling and manipulating his environment ... Chess is the intellectual game *par excellence*. Without a chance device to obscure the contest, it pits two intellects

against each other in a situation so complex that neither can hope to understand it completely, but sufficiently amenable to analysis that each can hope to outthink his opponent. The game is sufficiently deep and subtle in its implications ... to have allowed a deepening analysis through 200 years of intensive study and play without becoming exhausted or barren ... If one could devise a successful chess machine, one would seem to have penetrated to the core of human intellectual endeavour".

This position was supported and extended by Michie [66], who proposed that chess was useful for the study of knowledge representation in AI since:

- the game constitutes a fully defined and well–formalised domain;
- it challenges the highest levels of human intellectual capacity;
- the challenge extends over the full range of cognitive functions such as logical calculation, rote learning, concept–formation, analogical thinking, imagination, deductive and inductive reasoning;
- a massive and detailed corpus of chess knowledge has accumulated over the centuries in the form of chess instructional works and commentaries;
- a generally accepted numerical scale of performance is available in the form of the US Chess Federation and International ELO rating system.

In an interesting analysis A. Shapiro [123] lists five alternative domains against these criteria and found them all to fail on at least one. For example, analytical chemistry (successfully used in the DENDRAL project [52]) fails on the criterion of providing a fully defined and well–formalised domain. Again, school algebra could be used as an experimental domain, but fails on the second criterion, of challenging the highest levels of human intellectual capacity.

2.2 Previous work

In the previous section we mentioned the important rôle of chess as a domain for Artificial Intelligence research in general and Machine Learning in particular. We also noted in Chapter 1 the success in real world applications of Machine Learning methods originally developed and extensively tested in the laboratory on the problem of inducing classifiers for chess endgame positions. All the work referred to below is concerned, as is ours, with domain-independent general purpose inductive inference, despite the fact that chess endgames are the common experimental domain.

2.2.1 Initial results

In one of the earliest computer induction experiments in a chess domain Michalski and Negri [64] used the AQVAL/1 program in an attempt to induce a rule to classify positions in the KPK endgame as won or drawn. The rule induced from a sample of a few hundred selected training instances of "won" and "drawn" positions correctly classified 80% of a sample of 159 positions. The attributes used to describe the positions included the predicates devised in a previous attempt to program the KPK ending [130]. These were based on relations between the pieces such as relative distances between Kings and the Pawn. Other attributes using "high-level" chess knowledge were less important in the induced rules.

Experiments on a much larger scale were carried out by Quinlan [106] using his ID3 algorithm, variants of which were subsequently successful in a number of commercial applications [69, 71]. The task addressed by Quinlan was that of classifying positions from the KRKN chess endgame [47] as "lost n-ply" or not. The problem can be stated thus. For all legal black-to-move positions, does the Knight's side (i.e., Black) lose in at most n-ply ? A move by one side followed by a move from the opponent takes two ply. The definition of lost n-ply is recursive, as follows. A position is lost 0-ply if and only if Black is checkmated, or each of the following conditions is true: the Knight has been captured; the position

is not stalemate; and the Rook is still on the board and cannot be immediately captured. There are two cases for $n > 0$. Any white-to-move position (i.e., where n is odd) is lost n-ply iff there is a move for White giving a position which is lost $(n-1)$-ply. Any black-to-move position (i.e., n is even) is lost n-ply iff all possible moves for Black give positions which are lost $(n-1)$-ply.

The goal of Quinlan's study was to investigate the utility of learning decision-tree classifiers. He reported an experimental comparison with minimax and special-purpose search in the domain. The results clearly showed that the induced decision trees gave the most rapid method of classification for the cases of lost 2-ply and lost 3-ply. In relation to this thesis however it is the critical importance to the above results of the human-crafted attributes that is of interest. In Chapter 1 it was mentioned that the effort on behalf of the human programmer expended on devising an adequate set of attributes by trial-and-error in this work was very much greater than the time taken for ID3 to induce the final decision tree. The chosen attributes simplified the induction problem since many positions in the domain map onto one attribute combination, and most attribute combinations have no associated position. This simplification is evident from the following measures. A four-piece endgame such as KRKN has 64^4 or $16,777,216$ possible board configurations. By selecting only legal positions and ignoring those configurations which are equivalent due to symmetries this reduces to around 1.8 million positions for black-to-move and 1.4 million positions for white-to-move. Considering only the black-to-move positions lost 2-ply the figure reduces to $69,000$, and for white-to-move positions lost 3-ply to $474,000$. However, encoding the 1.8 million positions using 23 attributes for the lost 2-ply problem resulted in 428 example vectors. Similarly, encoding the 1.4 million positions using 39 attributes for the lost 3-ply problem resulted in 715 example vectors. The compression is dramatic, and reflects the knowledge encoded in the attributes, which for ID3 must be supplied by the user.

A number of studies comparing the performance of Michalski's AQ induction methodology with Quinlan's ID3 approach are in the literature [1, 91]. In

chess endgame domains the performance of both systems in these comparisons recorded differences in time taken for induction and the complexity of the induced rules. On the whole ID3 was found to be faster, although the ID3 decision trees were found to be more complex than the AQ-style production rules. No significant differences were recorded in predictive accuracy.

2.2.2 Structured induction

The issue of the complexity of induced rulesets becomes increasingly important as the domain size increases. Even in the restricted KRKN domain, Quinlan's ID3 experiments described above generated quite large trees. Specifically an 83 node tree was induced for the lost 2-ply problem, and a 177 node tree for the 3-ply problem. The application of these inductive techniques in larger domains is clearly likely to produce large trees which are usually difficult to comprehend.

The motivation behind the method of "structured induction" [122] was to extend the inductive approach to produce humanly-comprehensible decision trees, even for very large problems. This was achieved by having the expert decompose the problem into sub-problems as is done in the methodology of structured programming. The ID3 algorithm was then used to induce a decision rule for each sub-problem.

The problem decomposition was not a trivial task in the endgames chosen for a large-scale test of the structured induction approach, KPK and KPa7KR [123]. For KPK this took the major part of the 6 weeks of programmer time required for developing the structured solution. The corresponding time for the KPa7KR work was 10 weeks. It was further noted that by an information-theoretic analysis an estimated 84% of the classificatory power of the structured solution was accounted for by the expert-supplied structure.

Since the attributes for the induction within each sub-problem of the structured induction approach were developed top-down, simultaneously with the structure, and constrained by the requirement that they be evaluated on the

CLIP/C (a parallel array processor) emulator, it seems that the key to the structured induction method is the attribute design process.

2.2.3 Automating structured induction

In tackling the problem of inductively acquiring expert-level solutions for large problem spaces, structured induction introduced a new bottleneck, namely devising the structure. The automation of Shapiro's problem decomposition approach to induction was investigated using Michalski's statistical clustering algorithm CLUSTER in the KPK domain [96]. The motivation was to reduce the time taken to produce the structure within which decision trees were generated. However, the results were not successful, since the suggested structure was not recognisable to experts.

A different approach to automatic structuring resulted in Muggleton's Duce system [80]. The program uses six transformational operators which are applied to examples or rules to rewrite clauses containing common subsets of symbols. These operators implement both inductive and truth-preserving transformations. Duce uses propositional Horn clause logic to represent both rules and examples. It was subsequently shown that the Duce operators invert Robinson's resolution rule of deductive inference [83]. Semi-automated inductive structuring is achieved by the invention of new predicates in the course of inverting multiple resolution steps. The process may be fully automated, as it was in the version of Duce used in the experiments in this thesis, by disabling oracle queries.

Given the size of the potential search space in applying Duce to large sets of examples, it was important to test the program on a real problem. Muggleton chose the task of re-creating the structured solution in the KPa7KR domain [123] from "flattened" example vectors. With two chess experts acting as the oracle, the program induced a structured solution for the domain [80]. Although this solution was not quite as compact as the original, it was meaningful to experts,

guaranteed correct by construction, and required substantially less time to produce.

2.2.4 Attribute discovery

All of the work on the induction of decision rules in chess endgame domains cited above depends on the initial compression of the total space of positions using attributes devised by the programmer. In the case of the work on structured induction [123], these were developed as part of the top-down structuring process. This is apparent since three attributes for the KPa7KR domain were found to be redundant when an unstructured decision tree was induced. One of these was actually used redundantly in the structured solution. It is present as an artefact of the top-down development of the structure where an ID3 decision tree was generated independently for each sub-problem.

It was clearly apparent that despite the success of inductive methods in the endgame experiments the programmer, or knowledge engineer, was still being required to perform substantial "hand-crafting" in each of these applications. This recognition motivated the work in this thesis, which began with an attempt to tackle the attributes discovery problem in the domain of KRK Illegality using the Duce algorithm, as described in Chapter 3. We now proceed to define the KRK Illegality problem.

2.3 Database generation

Chess endgames are complex domains which are enumerable. Endgame databases are tables of stored game-theoretic values for the enumerated elements (legal positions) of the domain. The game-theoretic values stored denote whether or not positions are won for either side, or include also the depth of win (number of moves) assuming minimax-optimal play. From the point of view of experiments on computer induction such databases provide not only a source of examples

but also an oracle [118] for testing induced rules. However a chess endgame database differs from, say, a relational database containing details of parts and suppliers in the following important respect. The combinatorics of computing the required game-theoretic values for individual position entries independently would be prohibitive. Therefore all the database entries are generated in a single iterative process using the "standard backup" algorithm [132], described below.

2.3.1 Definitions

Chess endgames

An endgame is defined and described by the material on the board. By convention, endgames are described by listing the pieces for White's side in descending order of material value, followed by Black's pieces in the same order. Pieces are referred to by a single letter from the set {K, Q, R, B, N, P} (meaning king, queen, rook, bishop, knight, pawn). For example, KQPKQ denotes the five piece endgame with White possessing king, queen and pawn against Black's king and queen. There is a more compact encoding called the "GBR code" [117] proposed for the classification of chess games and subgames but this is as yet not widely used.

Sub-domains of endgames may be specified by constraining one or more pieces to be on a particular set of squares, e.g. the endgame KPa7KR includes all positions where White has king and a pawn fixed on square a7, against Black's king and rook. We use the convention that board positions are described in the algebraic notation.

Capturing a piece leads into a sub-game of the endgame, e.g. from KBNK into KBK when the black king takes the white knight. Additionally, there can be a pawn promotion relation between endgames, e.g. from KPK into KQK or KRK. These relations also apply between databases. Endgames related in this way are known as the derived endgames.

Since we will later be much concerned with the question of illegal positions,

there are two possible forms of illegality which must be mentioned. In both cases such positions are not illegal under a static evaluation of the inter-relationships of pieces on the board. However "orphan" or "unreachable" positions have no legal predecessors at some depth-level. Put another way, there is no way that one could move into these positions during normal forward play. The first case occurs when orphans are discovered and marked within the process of backing up while the database is being generated. This is handled by the standard backup algorithm described in Section 2.3.5.

There is also a second possibility for the occurrence of orphan positions in an endgame. These are positions which although legal within the endgame have no path of legal predecessors reaching back to the initial position of the full game with all 32 pieces on the board. These latter positions do not seem to be mentioned in the literature, apparently either on the assumption that the number of such positions involved in any case are negligible or that their occurrence is infeasible to check. In our consideration of the KRK domain we will ignore illegality due to either form of unreachability.

Endgame databases

A database is an ordered list of game-theoretic values for all positions in the endgame. For example, the truth value for the predicate "won-for-white with white-to-move" for every position could be stored. By convention, databases are normally constructed to store won-for-white values. There is a further convention identifying white with the "stronger" side. The stronger relation may not be evaluated in an obvious way. With bishop and pawn against rook, White's side might then be assigned the rook. Clearly both sides might have equivalent force, e.g. in KPKP.

Usually the depth of win is also recorded for each position. This allows the database to be used to play the endgame "optimally", using only a legal-move generator and a database look-up program. There is currently some discussion on alternative interpretations of optimal play in endgames [118]. In general,

however, play is described as optimal in the sense of the minimax [121] strategy.

All derived endgames should be referred to in constructing a database. This is to ensure that the correct minimax-optimal values are calculated for any positions in the database under construction which transform into other endgames by capture or promotion.

The size of databases is governed by the total number of piece combinations (positions) for the pieces on the board. An endgame contains on the order of 64^N positions, where N is the number of pieces in the ending. The combinatorial explosion makes an exhaustive enumeration impossible for endgames of more than a few pieces. A five piece endgame contains 64^5, or approximately 1 billion possible piece configurations. This includes many illegal positions such as those with one than one piece standing on the same square. Also included are positions equivalent through symmetry.

So far complete databases have only been built for endgames of up to five pieces [36]. Databases for endings with six or more pieces have been constructed by restricting the allowed positions, e.g. the KPPPKPPP [7] endgame, in which each side has three passed pawns.

To illustrate the complexities involved, we note the processing time for generation of two five-piece endgame databases (KBBKN and KQKBB). These were generated by K. Thompson at Bell Laboratories in the USA using a Sequent Balance 8000 parallel computer. He reported execution time of two to three weeks of real-time [132]. Each database requires over 121 MB of storage.

2.3.2 Symmetry

Reductions of roughly an order of magnitude in the size of an endgame domain are possible through various symmetries. The potential reduction is less if there are pawns in the endgame. Symmetry in the space of positions of an endgame is exploited in an attempt to minimise the storage size and generation complexity of a database for that endgame. Symmetries reduce the number of positions

in the database by storing only one value for a set of positions equivalent by combinations of reflections or rotations.

Pawn endgames have positions equivalent by reflection about an axis between files d and e. The so-called pure-piece endgames [132], i.e. those which do not contain pawns, have three axes of symmetry. We will not consider pawn endgames any further in this discussion.

In utilising symmetry for pure-piece endgames, the white king is restricted to the ten squares shown in Figure 7. Any position in such endgames can be reflected or rotated to place the white king in this area. For each of these squares the white king can occupy any square apart from those which place it on top of or adjacent to the black king. In addition, when the white king is on one of the squares of the diagonal $\{\mathrm{a1}, \mathrm{b2}, \mathrm{c3}, \mathrm{d4}\}$, then squares for the black king above this diagonal can be reflected to be below the diagonal. By removing combinations redundant through these symmetries, the total number of possible positions for two kings in pure-piece endgames is 462.

The effect of these symmetries is to ensure that a set of positions which are equivalent by certain reflections or rotations are represented only once in the database. This is usually referred to as the *canonical state-space* of the endgame.

Each value in the database has an address which is a function of the corresponding board position. This is the "access function", described further in Section 2.3.3. That function must include the transformation by symmetry of a position outside the canonical state-space to its equivalent within the state-space.

Note that there are other symmetries. Two white bishops could each be restricted to 32 squares of the board, which would reduce the storage requirement to 25% of the uncompressed size. There is further redundancy if there are two pieces P of the same colour in the endgame, since every pair of positions where P_1 and P_2 are interchangeable have the same game- theoretic value. In general, these are not frequently exploited, since the database access function then becomes more complicated and consequently slower.

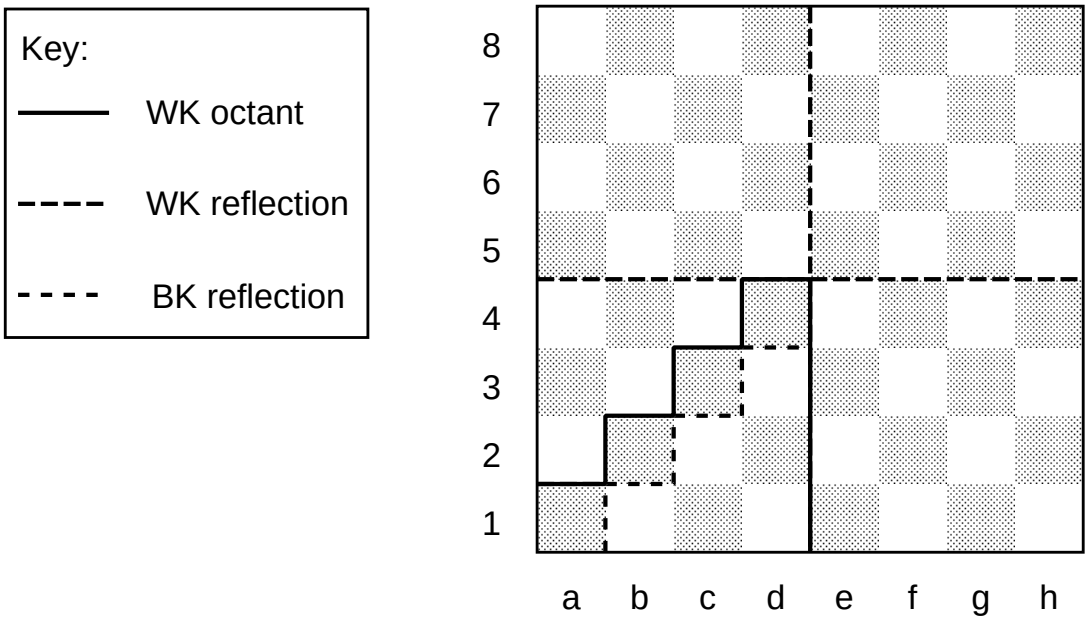


Figure 7: Canonical king positions for non-pawn endgames.

WK octant – the ten squares {a1, b1, c1, d1, b2, c2, d2, c3, d3, d4} are the canonical locations for the White King (WK).

WK reflection – reflection about the axes indicated places the WK in a canonical location:

> if the WK is above the central horizontal (rank 5 or greater),
> reflect the WK below (into the lower half of the board);
>
> if the WK is right of the central vertical (file e or greater),
> reflect the WK to the left of (into the left half of the board);
>
> if the WK is above the diagonal a1 to h8,
> reflect the WK below this diagonal (into the WK octant).

BK reflection – with the WK on squares a1, b2, c3 or d4 and the Black King (BK) above the diagonal a1 to h8, reflect the BK about this axis to place it below the diagonal.

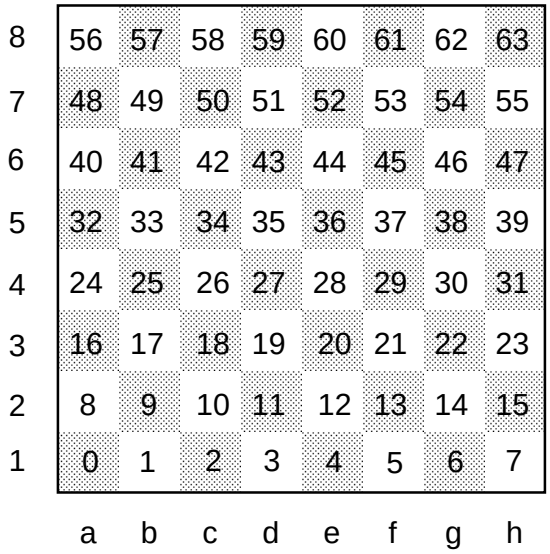


Figure 8: The numbering scheme for chess board squares used in the access function.

2.3.3 Access function

To build and use the database a fast and reversible access function is required. This function should return a database index given a position, or return a board position given a database index. The former is necessary when retrieving a value for a position in the database, the latter when generating the database or using it to play the endgame.

In the Section 2.3.2 the uses of symmetry were described and it was indicated that transformations due to symmetry are incorporated in the access function. However we will ignore it in our discussion of the access function to simplify presentation of the examples. The canonical representation of positions formed under symmetry excludes positions where Kings are on top of or adjacent to one another. As will be seen in Section 2.4 examples of these classes of positions are required in the induction experiments. It was also a goal at the outset of this work to investigate the inductive generalisation of rules of symmetry from examples, although this has not in fact been attempted.

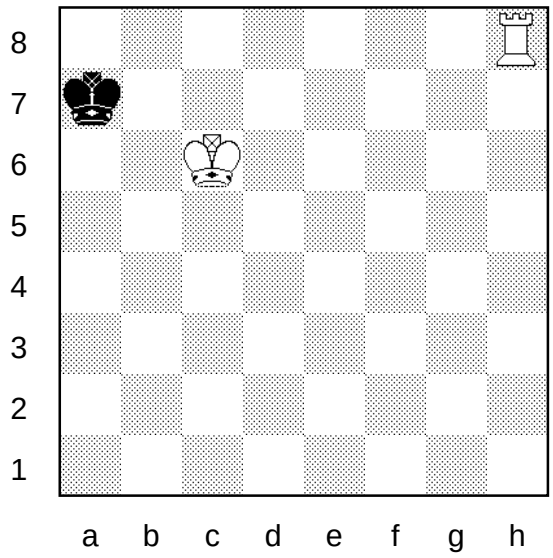


Figure 9: The KRK position WK:c6, WR:h8, BK:a7.

The access function is a total bijection or Gödel function [1] which maps positions to a subset of the set of natural numbers and that subset of the natural numbers to positions. In the following let

N = number of chess pieces in the endgame;

P = chess piece on a board square, $P \in \{0, \ldots, 63\}$, (see Figure 8);

$Position$ = N-tuple of P;

$Positions$ = set of all positions in the endgame;

$Max = |Positions| - 1 = 64^N - 1$,

$Index$ = database address;

$Indices = \{Index | Index \in Nat, 0 \leq Index < Max\}$.

First the forward function is defined.

$$lookup : Positions \mapsto Indices.$$

[1] The Gödel function should be thought of simply as an index.

Definition 2.1 (Access)

$$lookup(< P_1, P_2, \ldots, P_N >) = P_1 \times 64^{N-1} + P_2 \times 64^{N-2} + \cdots + P_N \times 64^{N-N}$$

For example, information concerning the position WK:c6, WR:h8, BK:a7 of Figure 9 would be represented in a hypothetical database at the address calculated as follows.

Example 2.2

$$lookup(< 42, 63, 48 >) = 42 \times 64^2 + 63 \times 64^1 + 48 \times 64^0 = 176112$$

Next the inverse function is defined.

$$lookup^{-1} : Indices \mapsto Positions$$

Definition 2.3 (Inverse Access)

$$\begin{aligned} lookup^{-1}(Index) \;=\; & < \lfloor \frac{Index}{64^{N-1}} \rfloor, \\ & (\lfloor \frac{Index}{64^{N-2}} \rfloor \bmod 64), \\ & \vdots \\ & (\lfloor \frac{Index}{64^{N-N}} \rfloor \bmod 64) > \end{aligned}$$

For example, the KRK position shown in Figure 9 and represented in the hypothetical database at the address 176112 would be derived as shown below.

Example 2.4

$$\begin{aligned} lookup^{-1}(176112) \;=\; & < \lfloor \frac{176112}{64^2} \rfloor = 42, \\ & (\lfloor \frac{176112}{64^1} \rfloor \bmod 64) = 63, \\ & (\lfloor \frac{176112}{64^0} \rfloor \bmod 64) = 48 > \end{aligned}$$

2.3.4 Initialisation

Database generation may be thought of as a two-stage process, sometimes termed "retrograde analysis" [132]. There is an analogy here with full-width search chess programs. The first stage is initialisation of terminal positions in the endgame, analogous to the evaluation function of a chess program for positions at the limit of its search depth. The second stage of database generation involves the propagation of game values from the terminal positions to all legal predecessors. This stage is analogous to the minimaxing carried out by a chess program. Note that the stages in each case are ordered in reverse. Alternatively, the processes of database generation and full-width search may be thought of as backward- and forward-chaining, respectively.

However, there is an important distinction between the largely heuristic evaluation of positions carried out in a chess-playing program, and the classification of terminal positions in database initialisation. The former only assigns an *approximation* to the game-theoretic value of a position, whereas the latter gives the exact value. For a given endgame, an initialisation predicate $Terminal(p)$ is true for a position p if and only if p is a legal mate for white at zero ply with black to move.

The actual initialisation consists of an iteration from 0 to N, the set of indices, where N is the number of entries in the database. The index for each database entry is converted to a board position by the inverse access function $lookup^{-1}$. This position is evaluated by the $Terminal$ predicate. The resultant truth value is then entered in the database file "black.db" at an address given by the index. Following initialisation of all black-to-move terminal positions, the database file "white.db" which is to contain values for all positions in the endgame with white-to-move is initialised with all entries set to zero. Note that files for both black- and white-to-move are required for the storage of intermediate values during backing up, although the final database will only contain values for white-to-move.

2.3.5 Backing up

The backing-up stage entails the propagation of game values back from the initialised positions until all "reachable" positions in the endgame are assigned values for minimax-optimal depth of win. By reachable is meant positions which are legal predecessors of the initialised positions or their legal predecessors. Predecessors are generated by making "un-moves" [132]. The rules for un-moves are the same as for moves, except that it is illegal to un-move out of check, but legal to un-move into check. Also, it is illegal to capture, but legal to un-capture by "uncovering" an opponent's piece. A complete legal un-move generator would additionally include un-castling, un-*enpassant* and un-promotion for pawn endgames. Successors of positions are generated using a legal-move generator.

The procedure given below is in outline the "standard backup" algorithm for generating a chess endgame database. Several more detailed accounts of different implementations appear in the literature, e.g. [123, 132, 134].

The first stage is initialisation, described in Section 2.3.4. Following the initialisation stage a counter d, for depth of win in ply, is set to zero. The backup algorithm then repeats steps 1 to 3 below until no new database entries are made during a complete iteration.

1. For each black-to-move position entered in "black.db" as lost at depth d:
   (a) Generate the set of all predecessor (white-to-move) positions.
   (b) If this set is empty, mark the black-to-move position in "black.db" as unreachable.
   (c) Otherwise, if these predecessors are not already entered in "white.db" as won at depth $\leq d$, then enter them in "white.db" as won at depth $d + 1$.
2. For each white-to-move position entered in "white.db" as won at depth d:

(a) Generate all predecessor (black-to-move) positions.

(b) If this set is empty, mark the white-to-move position in "white.db" as unreachable.

(c) Otherwise,

i. form a list containing each of these black-to-move predecessors which are not already entered in "black.db" as lost at depth $\leq d$.

ii. for each position in this list, generate all legal white-to-move successors.

iii. if all the white-to-move successors to this black-to-move position are entered in "white.db" as won at depth $\leq d$ then enter the black-to-move position as lost at depth $d + 1$ in "black.db".

3. Set $d := d + 1$.

When the program terminates, the file "white.db" holds the values for the completed white-to-move database.

Note that there is a large amount of database access to be done during backing-up. This is because at each ply very many positions must be checked before game values can be assigned. The practical consequence of this is that a computer with large core memory is required if the program is not to become disk-bound.

2.3.6 A KRK database

The particular endgame used as an experimental domain in this work was King and Rook against King (KRK). This endgame was originally chosen since it is known to be "one of the easiest of the standard mates" [65]. Black cannot win and is unable to draw at depths greater than zero. Table 1 shows the depth-of-win in moves for KRK positions with black- and white-to-move. The latter was used to classify example positions in our induction experiments. Both databases were generated using a version of the standard backup algorithm in

Table 1: Tabulation of positions in the KRK databases.

depth	black-to-move	white-to-m ove	
0	27	189	
1	78	587	
2	246	484	
3	81	238	
4	198	607	
5	471	1091	
6	592	1418	
7	683	2149	
8	1433	2514	
9	1712	2382	
10	1985	2565	
11	2854	2691	
12	3597	2234	
13	4194	2027	
14	4553	662	
15	2166	121	
16	390	0	
total	25260	21959	
drawn	2796	6097	check
illegal	1512	1512	
grand total	29568	29568	

Section 2.3.5, implemented at the Turing Institute by Arthur van Hoff supervised by the author. The databases each contain $462 \times 64 = 29568$ entries.

One advantage of working with the KRK endgame is that a previous tabulation appears in an appendix to a paper in the literature by Clarke [20]. This allowed some checks on our database to be performed by comparison with the previous work. From this comparison it was found that both tabulations agreed on all numbers of won and drawn positions at each depth. In Clarke's case both databases contained 32896 entries, since although symmetry was applied to restrict the black king to ten squares of the board, illegal combinations of both kings were not excluded from this "canonical representation". He also gave separate figures for black-to-move stalemates and draws with rook *en prise*. There is an apparent discrepancy in his printed table where the total number of black-to-move losses is given as 25233. This seems to omit the 27 depth 0 losses where black is mated, and should read 25260 as in our Table 1.

2.4 Reconstruction of KRK Illegality

As mentioned above the KRK endgame is one of the easiest of the standard mates. This made it an ideal domain for the investigation of induction from primitive data. The target concept chosen for our experiments was the classification of KRK positions as illegal or not. KRK is relatively simple in chess terms and has been the subject of a number of mechanized solutions [65]. Despite this, even the problem of inducing a set of classification rules for illegality turned out to be sufficiently taxing for our purposes, as will be seen in later chapters.

The general framework chosen for the testing of induction methods on this laboratory problem was the reconstruction experiment. With this design a known target theory is required. Instances of this theory form the example sets supplied to the agents whose learning powers are to be investigated. The learning task for an agent is to reconstruct this theory from the examples. Success on this task is judged according to criteria supplied by the experimenter.

2.4.1 A target theory of KRK illegality

The classification of white-to-move KRK positions as illegal by our target theory was dependent only on static evaluation of the relationships between the pieces on the board. No concept of moving pieces by transforming board positions was involved. The possibility of piece configurations being illegal due to "unreachability" as discussed in Section 2.3.4 was ignored. Positions were represented by a 6-tuple the arguments of which denote the file and rank values (x and y coordinates) of, in order, the White King, the White Rook and the Black King. This representation was chosen to correspond with the algebraic notation used by human chess players.

As presented in Figure 10 three intermediate-level concepts define sub-classes of KRK illegal positions. The top-level predicate in the Prolog [21] program is "illegal/6" which calls three predicates for each of the three sub-classes. The first of these, "same_sq/6" is true for any positions in which two or three pieces have the same coordinates, i.e. occupy the same board square. In the second, "adj_kings/6", if the two kings are on diagonally, vertically or horizontally abutting squares then the position is illegal. The third predicate "bk_check" is true for positions in which the White Rook is attacking the Black King and the White King is not blocking the attack. This sub-class of positions is illegal for white-to-move since it implies that on the previous play the Black King moved into check. The predicate "inbetween/6" defines the blocking relation in positions where all three pieces line up on the same rank or file (row or column).

This program also requires the utility predicates defined for the 8×8 board. The "succ/2" successor predicate is true when the first argument is a file (rank) adjacent to the file (rank) of the second argument. The "lt/2" predicate is that of strictly-less-than ($<$) for files or ranks. With separate domains for file and rank values (respectively $\{a, \ldots, h\}$ and $\{1, \ldots, 8\}$) the definition of these utility predicates consisted of 70 atomic clauses.

```
% KRK illegality
illegal(WKf,WKr,WRf,WRr,BKf,BKr) :-
   ( same_sq(WKf,WKr,WRf,WRr,BKf,BKr);
     adj_kings(WKf,WKr,WRf,WRr,BKf,BKr);
     bk_check(WKf,WKr,WRf,WRr,BKf,BKr) ).

% Same square
same_sq(A,B,A,B,_,_).
same_sq(A,B,_,_,A,B).
same_sq(_,_,A,B,A,B).

% Adjacent kings
adj_kings(A,B,_,_,A,C) :- succ(B,C).
adj_kings(A,B,_,_,A,C) :- succ(C,B).
adj_kings(A,B,_,_,C,B) :- succ(A,C).
adj_kings(A,B,_,_,C,B) :- succ(C,A).

adj_kings(A,B,_,_,C,D) :- succ(A,C), succ(B,D).
adj_kings(A,B,_,_,C,D) :- succ(A,C), succ(D,B).
adj_kings(A,B,_,_,C,D) :- succ(C,A), succ(B,D).
adj_kings(A,B,_,_,C,D) :- succ(C,A), succ(D,B).

% Black king in check
bk_check(A,B,C,D,C,E) :- not(inbetween(A,B,C,D,C,E)).
bk_check(A,B,C,D,E,D) :- not(inbetween(A,B,C,D,E,D)).

inbetween(A,B,A,C,A,D) :- lt(B,C), lt(D,B). % Same file,
inbetween(A,B,A,C,A,D) :- lt(C,B), lt(B,D). % WK blocks.

inbetween(A,B,C,B,D,B) :- lt(A,C), lt(D,A). % Same rank,
inbetween(A,B,C,B,D,B) :- lt(C,A), lt(A,D). % WK blocks.


70 additional background predicates: succ/2 (14), lt/2 (56).
```

Figure 10: A "target theory" of KRK illegality in Prolog.

2.4.2 An enumeration of illegal KRK positions

Supervised learning is the category into which all agents tested in our experiments would be placed. The definition of this category covers all learners which rely on some kind of teacher to label training examples according to their concept class. In most of our experiments the example positions were randomly generated then classified as illegal or not by accessing the KRK database. The white-to-move positions illegal due to the Black king being in check are indicated separately in the database tabulation of Figure 1. This number of 6097 together with the other 1512 illegal positions in the database is with symmetries removed.

Randomly generated KRK positions may be thought of simply as the result of dropping the three pieces on any of the 64 board squares. Under this scheme the total number of different three-piece configurations is $64^3 = 262144$. A complete enumeration of these possible configurations was performed, and each was classified using the target program described above. The total numbers of legal and illegal positions are given in Figure 11. From this we can see that the ratio of legal to illegal positions is almost exactly $2 : 1$. In addition, of the different sub-classes of illegal positions, there are only around half as many cases of "king adjacency" as for "Black King in check". Again, there only half as many positions with pieces on the same square as for king adjacency.

Figure 11 also gives the sizes of intersections between the different sub-classes of illegal positions. These are useful in allowing calculation of the effect of execution order for different sub-classifications. If an illegality classifier program with separate procedures for deciding each sub-class is thought of a "filter" then the order in which these procedures is executed has an effect on the proportions of positions falling into each sub-class. Imagine that two programmers write filter programs with semantics equivalent to that of our target theory. The first programmer is a chess player, and orders the sub-class decision procedures to check first for pieces on the same square, then king adjacency and lastly Black

Total number of positions : 262144

No. of legal positions :	175168 (66.8%)
No. of illegal positions :	86976 (33.2%)
same square (S) :	12160 (4.6%)
kings adjacent (A) :	26880 (10.3%)
BK in check (C) :	59648 (22.7%)
$S \wedge A$ (SA) :	840
$S \wedge C$ (SC) :	5888
$A \wedge C$ (AC) :	5628
$S \wedge A \wedge C$ (SAC) :	644

"Filter" order:

(1) S, A, C (chess):

$S = 12160$ (4.6%)
$A - SA - SAC = 26880 - 840 - 644 = 25396$ (9.7%)
$C - SC - AC - SAC = 59648 - 5888 - 5628 - 644 = 47488$ (18.1%)

$12160 + 26040 + 48776 = 86976$ (33.2%)

(2) C, A, S (random sampling) :

$C = 59648$ (22.7%)
$A - AC - SAC = 26880 - 5628 - 644 = 20608$ (7.9%)
$S - SA - SC - SAC = 12160 - 840 - 5888 - 644 = 4788$ (1.8%)

$59648 + 21252 + 6076 = 86976$ (33.2%)

Figure 11: Classification of KRK positions into illegality sub-concepts by target theory.

King in check. This order could be based on ease of encoding each procedure, or how obviously distant from a normal chess game would be any position in each sub-class. This forms case (1) in Figure 11. Case (2) corresponds to the filter ordered by a programmer with no interest in ordering of the procedures to detect the three sub-classes of illegal positions except for the frequency in which they occur under random sampling. For efficiency this programmer decides to check for the occurrence of the most common sub-class first, and so on. It is apparent that this sub-class ordering is reversed. We shall see in later chapter that these sub-class intersections can have an effect on the inductive reconstruction of the target theory.

2.5 Summary

In this chapter the experimental domain to be used throughout the thesis has been described. The basis of the database generation technique for chess endgames was outlined and the top-level algorithm given. The technique has become standard even though it has in the main been applied only to endgames with a small number of pieces on the board. Results were given for the KRK endgame database used to classify the legal and illegal positions comprising some of the example sets in our induction experiments. A summary of depth-of-win entries in the database was also given. Samples of positions classified from the database in this way are used as experimental materials in Chapter 7.

Chapter 3

Induction of attributes

The starting point for the work presented in this thesis was the observation that decision-tree induction [107, 11] involves the construction of classification rules for concepts in terms of "attributes" and that these descriptive features, which typically contain a significant amount of domain knowledge, must be supplied by the user. This has been reported by the authors of several of the most successful applications of this type of Machine Learning [106, 123, 23]. In these cases the time taken to devise an adequate set of attributes for the problem has been a significant proportion of the total time spent in completing the application.

A typical formulation of the learning task, for decision-tree induction and other algorithms, requires a set of examples from the domain. These are described in terms of their class membership and attribute values. To induce rules for class membership in terms of the attributes and their values, the examples must be described in terms of an adequate set of attributes. A set of attributes is adequate when no examples from different classes have the same set of attribute-value pairs [105].

The attributes may be either "observational" or "theoretical terms". Observational terms appear in the language of observations for a given domain. In physical domains they correspond to measurement variables, otherwise they are given by definition. In either case they constitute the primitive data, i.e. the

lowest-level domain information obtainable. Theoretical terms are those which do not appear in the language of observations, but may appear in the hypothesis language. In our Figure 1 in Chapter 1, the level of primitive data corresponds to the observational terms, while the level of intermediate descriptions corresponds to the theoretical terms. Induced rules or decision trees, etc. correspond to the descriptive rules at the top of Figure 1.

For the present work we define "primitives" to be those properties of a domain which cannot be completely described in terms of any other known properties of that domain. In a grammar for a language of domain concepts they would be the terminal symbols.

To reiterate, primitives are given by definition in model or artificial domains, as in chess with the board and pieces. In real-world or physical domains, the primitives are those available from the sensory information at highest resolution. This is diagrammed in Figure 12.

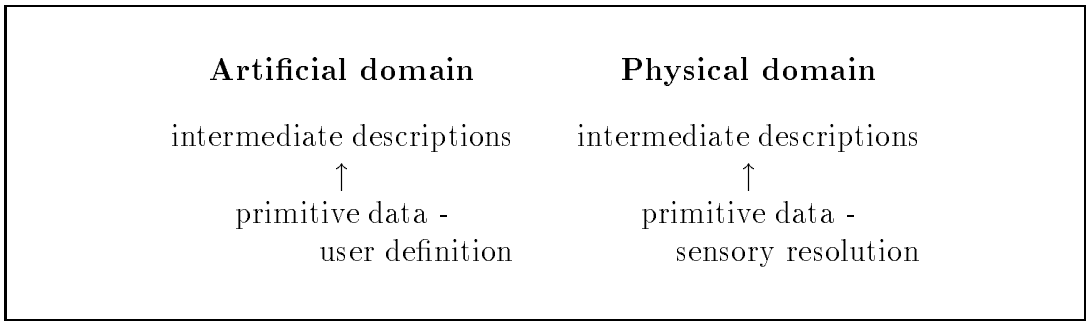


Figure 12: Attributes defined in terms of primitives in different domains. Most work on inductive learning has been concerned with the formation of high-level concepts or rules from intermediate descriptions or attributes. The attributes are typically assumed to be given. We are concerned with the induction of such attributes from primitives.

The problem of automating the provision of attributes for the induction of concepts can then be re-formulated as the induction of attributes from domain primitives.

In Section 3.1 of this chapter we consider general aspects of the attributes induction problem in terms of the adequacy of primitives and the compressibility

of expressions of primitive facts. The rôle of compression, or symbol reduction, in the action of inductive operators applied by the DUCE and CIGOL systems was the starting point for the exploratory experiments reported in Sections 3.2 and 3.3. Following from these initial trials an experimental framework for the studies reported in the later chapters of this thesis was developed, and this is presented in Section 3.4.

3.1 Adequacy and compression

Machine Learning algorithms can be classified on the basis of their representations, for example those using 0th-order languages (decision trees, propositional calculus rules, etc. which are formally equivalent [13]), compared with those using 1st-order representations (usually variants of the predicate calculus, as in [63, 119, 86]).

Induction algorithms using propositional representation languages (such as ID3 [107], AQ11 [62], DUCE [80]) require as input on a given problem a training set of example instances each with a known class value and a vector of values for a given set of descriptive attributes, e.g. expressed as a propositional Horn clause :

$$\text{Class\_Value} \leftarrow \text{Attribute}_1\text{\_Value} \wedge \text{Attribute}_2\text{\_Value} \wedge \ldots \wedge \text{Attribute}_N\text{\_Value}.$$

A subset of the primitive attributes available in a particular domain is said to constitute a distinguishing or *adequate* set of attributes for a training set of example instances if there are no instances with different class values which take the same set of attribute values [105].

In any domain, the maximum number of predicates, or the set of all subsets, is 2^N where N is the number of objects in the domain. This clearly leads to a very large number of predicates as N increases. One of those predicates is true and one is false for all objects in the domain. These represent, respectively, the set of all domain objects and the empty set.

For any induction task, the possible classes and attributes display a many-to-one mapping to the set of predicates over a particular set of objects. A class may be thought of simply as another attribute.

Each possible attribute is a member of an equivalence class of possible attributes. Each equivalence class corresponds to a predicate in the domain. Only one attribute from each equivalence class is relevant to a particular classification task in the sense that the addition of any other attribute(s) from that equivalence class to an attributes set cannot affect the classificatory power of that attributes set.

If one attribute is arbitrarily selected as the class or target predicate, then there will be a conjunction of relevant attributes or their negations which defines the same set of objects as that target predicate and is in some sense a minimum distinguishing set of attributes.

However, why use many attributes to define the same set of objects as that defined by the target predicate ? There are at least two reasons. Firstly cost, or computational efficiency, since one goal of induction is to find an adequate set of attributes which is cheaper to evaluate than the target predicate. Second is the degree of human understandability of different attribute combinations or concept descriptions relative to the target predicate. This appears to relate to the complexity or size of the concept description, which is relative to the size of the set of learning examples should exhibit some degree of *compression*.

Note that these issues may be treated separately from estimation of classification error, the most usual performance criterion for measuring improvement through learning. Breiman et al. [11] have demonstrated that smaller, more comprehensible trees tend to be more accurate classifiers.

3.1.1 Adequacy

In artificial domains like chess example material can be generated from a complete specification. As discussed in Chapter 2, in chess endgames a complete

enumeration of positions is possible, within machine limits. In such an exhaustive enumeration each position is described using a unique index value, (Gödel number), and labelled with the value of its database classification. Therefore, it could be argued that Gödel numbers provide an adequate set of attributes. Similarly, in the case of KRK illegality the piece coordinate primitives used are adequate since positions are described uniquely and no position is classified as both legal and illegal.

However, the material contained in a database, from which example position descriptions can be generated, is not thought to be expressed in a representation which will facilitate learning the domain concepts of interest. Nonetheless, in the database each position is classified and described using an adequate attribute set, by definition. Is it therefore possible to do any induction at such a low level of description ? This is the question we set out to answer in this work.

3.1.2 Compression

One common method of comparing candidate hypotheses is based on a simplicity ordering [2]. An advantage of an ordering based on simplicity, or size, of hypotheses is that it is sample independent. With reference again to the extensional definition of a chess endgame constituted by its exhaustive enumeration as stored in a database, we can imagine many simpler (more compact) definitions with the equivalent denotation. For example, an algorithm to generate the database, together with its required inputs, could be one such (intensional) definition. The process of induction may be viewed as hypothesising such compressed definitions (e.g. a set of rules) given some part of a larger definition (e.g. a subset of the complete extension).

This is similar to the informal principle of scientific inquiry known as Occam's Razor, which states that the simplest hypothesis consistent with the data is to be preferred. The same principle is the basis of the inductive search for

"good" applications of the operators in the Duce algorithm [80]. (In fact, Occam's Razor has excited interest in several areas of Machine Learning, e.g. [8]). Taking this together with the fact that Duce also uses transformational operators to invent new predicates not present in the measurement data (i.e. theoretical terms) suggested the initial hypothesis that Duce would be suitable for the induction from primitives of sub-concepts (attributes) for KRK illegality.

3.2 An exploratory experiment in ZOL

As an initial investigation, the task reported here was to learn concepts relevant to classifying white-to-move KRK positions as legal or illegal by running Duce on sets of example positions chosen at random from the total state space of the endgame.

3.2.1 Description language for examples

At the outset it was necessary to choose a method of describing board positions in a $0th$-order (propositional) language of attribute-value pairs. In this section we discuss an experiment to test if descriptions using only two attributes for each piece would be sufficient to discover any structure relevant to the concepts used in the original position classification. This language of coordinates was chosen as primitive, in the following sense.

Since the domain is artificial, a number of representations could be primitive. However, considering the human chess player, the lowest-level descriptions of positions use tuples of (x, y) coordinates for location of pieces in the standard algebraic notation. Consequently, positions were described by giving two attributes for each piece, the *file* and *rank*, the former having a value in the range a to h and the latter having a value in the range 1 to 8.

Example positions were classified as legal or illegal by looking up their game-theoretic values in a database for KRK with white to move. As discussed

in Chapter 2, when generating this database a position was classified as being illegal if any of the pieces occupied the same square, or if the two kings were next to each other, or if the black king was in check. The possibility of a position being illegal because it has no predecessor positions which can be attained by a legal backwards move was ignored. All other positions are legal.

The first step was to take a very small sample of positions from the KRK database and run Duce on these examples, to test if any meaningful transformations were suggested. The ten examples used are shown below. Of these randomly selected positions, four were classed as illegal. In one of the illegal positions the black king and the white rook were on the same square. In the other three, the black king was in check to the white rook, which is illegal for a position with white to move.

```
(['-' legal] ([bkfile d] [bkrank 1] [wrfile d] [wrrank 3]
              [wkfile a] [wkrank 7])
(['-' legal] ([bkfile b] [bkrank 3] [wrfile g] [wrrank 3]
              [wkfile g] [wkrank 7])
(['-' legal] ([bkfile b] [bkrank 6] [wrfile b] [wrrank 2]
              [wkfile d] [wkrank 5])
(['-' legal] ([bkfile g] [bkrank 7] [wrfile g] [wrrank 7]
              [wkfile f] [wkrank 6])
```

The six legal positions are shown below. Generalisations of legal position examples were excluded.

```
(legal ([bkfile f] [bkrank 5] [wrfile d] [wrrank 2] [wkfile d] [wkrank 4])
(legal ([bkfile g] [bkrank 6] [wrfile f] [wrrank 7] [wkfile c] [wkrank 4])
(legal ([bkfile f] [bkrank 5] [wrfile d] [wrrank 2] [wkfile f] [wkrank 2])
(legal ([bkfile c] [bkrank 7] [wrfile e] [wrrank 2] [wkfile f] [wkrank 4])
(legal ([bkfile c] [bkrank 3] [wrfile b] [wrrank 5] [wkfile e] [wkrank 3])
(legal ([bkfile b] [bkrank 7] [wrfile d] [wrrank 6] [wkfile c] [wkrank 1])
```

The illegal example positions are shown above as rules with "['-' legal]" on the left hand side. Duce applied the generalisation operator TRUNCATION to each of these to produce the new rules for the concept "illegal" shown below.

```
(['-' legal] ([bkfile d] [wrfile d])
(['-' legal] ([bkrank 3] [wrrank 3])
(['-' legal] ([bkfile b] [wrfile b])
(['-' legal] ([bkfile g] [wrfile g])
```

In each case the generalisation relates to the concept of "black king in check", although the preconditions in the above rules are too weak to correctly express this concept. This is because the concept of "black king in check" is only fully specified if the black king and white rook are on the same file or rank, given the additional constraint that the white king does not block the attack. Also, in the fourth example, the generalisation to a "black king in check" type of rule could have been "black king and white rook are on the same square". This would be specified by :

```
(['-' legal] ([bkfile g] [bkrank 7] [wrfile g] [wrrank 7])
```

Note that generalising the fourth example to the rule that the position is illegal because the black king is in check when the white rook is on the same *file* with no other pieces blocking the attack :

```
(['-' legal] ([bkfile g] [wrfile g])
```

or that the black king is in check when the white rook is on the same *rank* with no other pieces blocking the attack :

```
(['-' legal] ([bkrank 7] [wrrank 7])
```

results in *greater symbol reduction* than specifying that the position is illegal because the two pieces occupy the same square. To say that the position is illegal because the pieces are on the same square perhaps makes more sense for this example position than saying that the position is illegal because the king is

attacked. However, since the latter concept is more concise, it will be suggested by Duce's simplicity ordering heuristics before the former.

At this point we may conclude that describing positions using attributes which refer only to the file and rank of each piece is at least powerful enough to permit Duce to discover some meaningful structure even with very few examples. However, with the above example set many potential transformations of approximately equal symbol reduction were investigated before those shown above. This is because there are too few examples for large regularities present in the rule base to constrain the search. Increasing the number of examples to 100 did not solve this problem, since a large number of irrelevant questions were still being asked of the human oracle. To increase the number of symbols in the rule base, either the primitive attributes should be described using more symbols, or the number of examples should be increased. The first option was rejected in favour of examining the effects of running Duce on sets of large numbers of examples.

3.2.2 Partitioning into sub-domains

Continuing this exploratory experiment, further trials were run on samples of 1000, 5000 and 10000 examples. The conclusions from these trials, which will be referred to as KRK1000, KRK5000 and KRK10000 for convenience, are summarised below.

In the KRK1000 trial, by accepting an initial transformation which partitioned the domain by fixing one attribute value, the search was subsequently constrained to investigate further partitions where each value of the same attribute was fixed. In all cases, the partitioning operator was DICHOTOMISATION, which generalises over sets of rules with positive and negative heads. For example, the initial transformation in KRK1000 is reproduced in this partial trace :

```
!- induce
Node= () V= 4 Op= Dichot Mem= 274933
Node= ([wrfile f]) V= -144 Op= Dichot Mem= 293765
DICHOTOMISATION
Rules:
        (legal ([wrfile f] '?'))
DICHOTOMISATION
Rules:
        (['-' legal] ([wrfile f] ['-' '?']))
What shall I call <?>? (name/n/i) ['-' illegalkrfk]
```

and the next question was

```
!- induce
Node= () V= 998 Op= Inter Mem= 307469
Node= ([wrfile d]) V= -127 Op= Dichot Mem= 328685
DICHOTOMISATION
Rules:
        (legal ([wrfile d] '?'))
DICHOTOMISATION
Rules:
        (['-' legal] ([wrfile d] ['-' '?']))
What shall I call <?>? (name/n/i) ['-' illegalkrdk]
```

Note that for the first two suggested transformations there is no extra search than that shown above. In fact, the search proceeds to partition the space of legal and illegal positions into eight sub-domains. Each sub-domain contains all legal and illegal positions where one value of the initially chosen attribute is fixed. The sub-domains were named by insertion of letters or numbers at

the appropriate position in the symbol "illegal-krk" to denote the file or rank attributes fixed in the particular sub-domain (see Figure 13). In this case, each sub-domain has only positions in which the white rook is on a particular file. In KRK1000, each sub-domain contains an average of $\frac{1}{8}$ of the total positions in the original sample. A diagram of the rule hierarchy following the partition into eight white-rook-file sub-domains is in Figure 13. However, after this partition into sub-domains, no further meaningful structure was found, and this trial was terminated in order to try larger example sets.

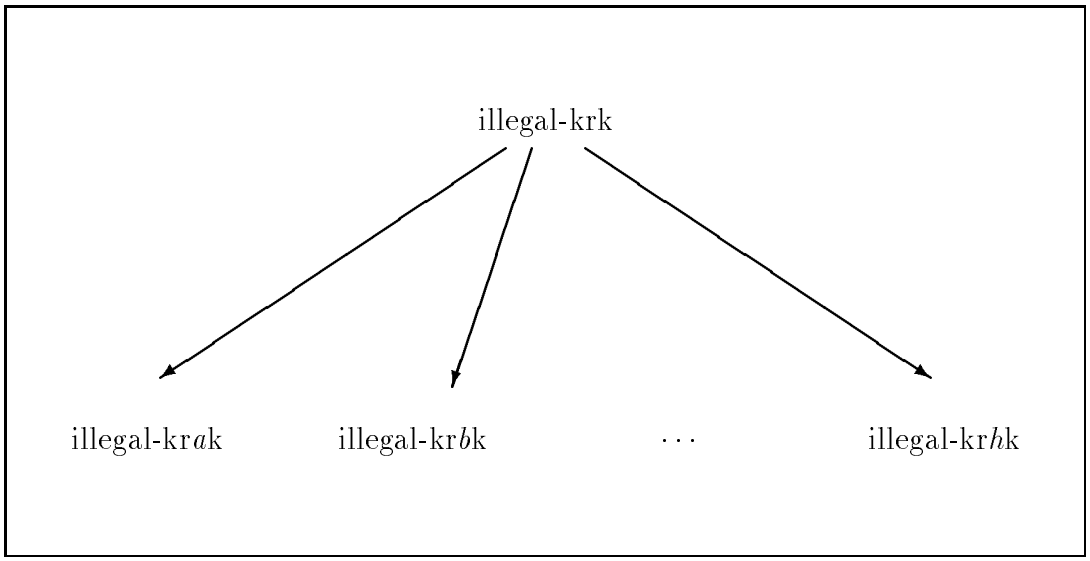


Figure 13: Sub-domain partitions in KRK1000.

In the KRK5000 experiment the initial transformation chosen was a partition on the attribute 'bkfile'. The choice of a file attribute was arbitrary, but the black king was chosen since it features in both the "black king in check" and "adjacent kings" concepts. The partitioning into sub-domains happened as before, and was repeated in the largest sub-domain illegal-krk$c$ by partitioning into sub-sub-domains on the attribute 'wrfile'.

An attempt to structure the next-largest sub-domain illegal-krk$g$ failed after partitioning into four sub-sub-domains on the attribute 'wrfile'. The reason appears to be that too few examples remained in the sub-domain to enable the induction of a complete second-level partition. This session was then terminated.

The resulting structure is diagrammed in Figure 14.

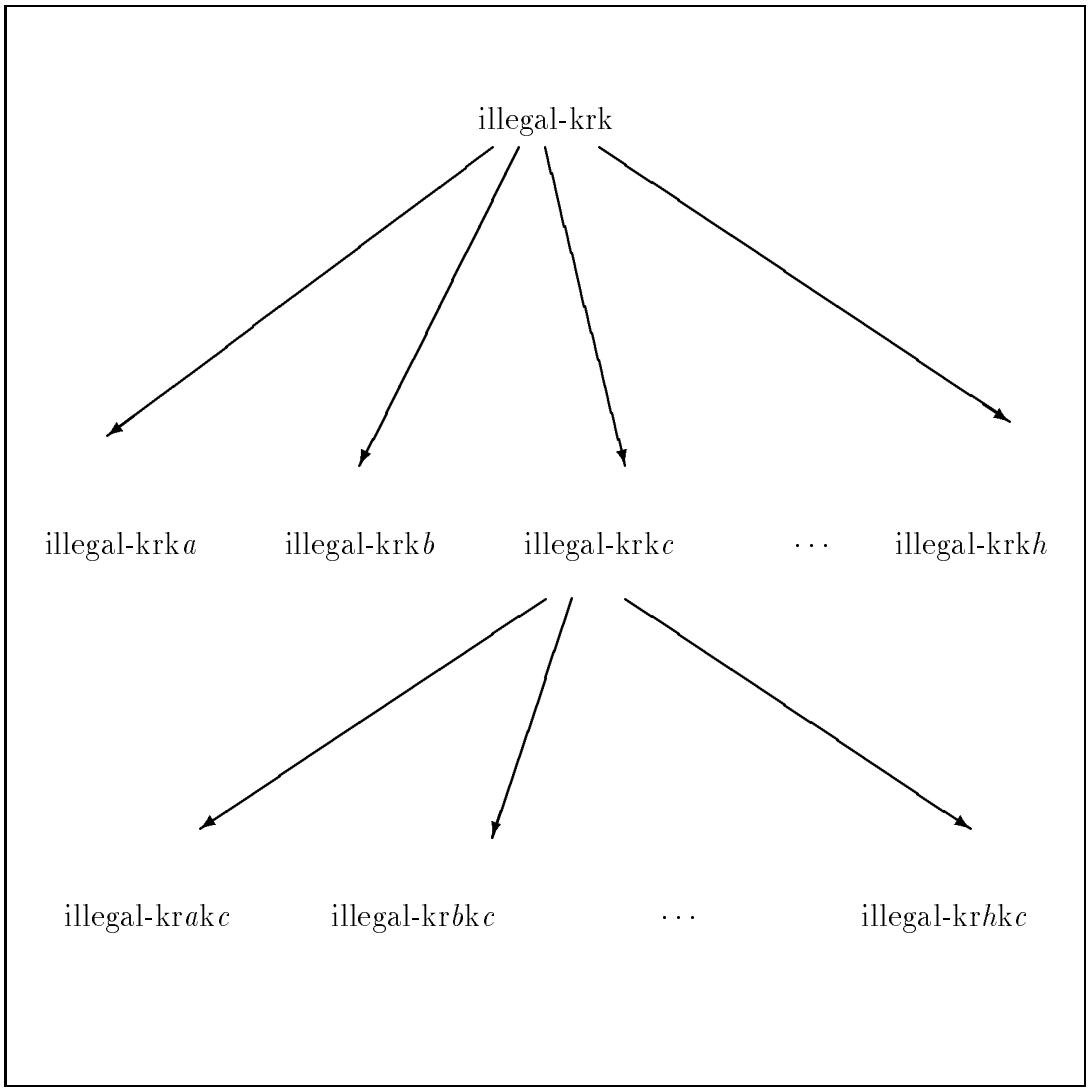


Figure 14: Sub-sub-domain partitions in KRK5000.

A sample of 10000 example positions was generated. This was used in a trial referred to as KRK10000. Again, the program proceeded to structure the domain by partitioning on the initially chosen attribute. Unfortunately, this sample proved to be too large for the machine being used, an 8MB Sun 3/75 workstation, and the session had to be abandoned.

In summary, from the exploratory experiment using Duce the following conclusions regarding induction of attributes may be drawn. The main favourable

result was that some meaningful structure for the target concept of KRK illegality was induced. This is evident in Figures 13 and 14. It is also clear that this structure partitions the training sample to express the sub-concept of BK in check. In the hypothesis language of primitives, however, this appears as an enumeration of the values for the attribute on which the partition is based. Although new intermediate concepts were invented following the application of the DICHOTOMISATION operator to enable partitioning, this "enumeration problem" prevented the partitioning from leading to a concise description of the target concept. We note that this effect is a consequence of a deficiency of propositional hypothesis languages, and has indeed been observed (see below) with other algorithms using similar representations, such as ID3. This observation lead to the comparative study of Chapter 4. A related deficiency of propositional hypothesis languages was noted by Pagallo [95] and termed the "replication problem".

Taken together, this suggested that a hypothesis language restricted to propositional expressions was not sufficiently powerful to express the target concepts concisely for this problem. Confirmation that the defect is indeed to be sought in this restricted expressivity of the hypothesis language is contributed by a recent study [73] using the same KRK illegality domain. In this study it was found that a state-of-the-art decision-tree learner, when presented with essentially the same task as Duce, also failed to improve on a form of random guessing. When supplied with additional attributes compounded pairwise from the six primitives, decision-tree learning rose to near 100% accuracy while remaining unsatisfactory in terms of intelligibility of the induced trees.

3.3 An exploratory experiment in FOL

In this work we move to induction in a more powerful hypothesis language, First Order Logic (FOL). More specifically, CIGOL [86] was used to learn Prolog clauses from ground examples of illegality. This move to FOL with an algorithm

which implements induction by Inverse Resolution leads to an increase in the total size of the hypothesis space compared with that of Duce for the same examples in the KRK domain. However, we know that our target concept of illegality is concisely expressible in Prolog (see Figure 10 in Chapter 2). As noted by Muggleton [81]

> By definition, inductive construction of first order theories involves a form of information compression. This follows from the fact that most finitely expressible first order theories entail an unbounded set of instances.

In common with Duce, CIGOL has the ability to invent new predicates by the inversion of multiple resolution steps. Bearing in mind that in Section 3.2 our negative result was mainly due to an inadequately expressive hypothesis language, we carried out some preliminary work with tutorial example sets. To show the operation of CIGOL, we now provide a worked example of sub-concept induction.

3.3.1 Example of sub-concept induction in CIGOL

To illustrate some steps in the process of induction by inverse resolution [83], Figure 15 shows part of a logged session with CIGOL. In this extract a new relation, a two-place predicate which the user names **adjacent_ranks**, is invented from raw data [1]. The remainder of this section provides some explanation of Figure 15.

In Figure 15 all CIGOL output is shown in boldface type, while all user input is in italics. CIGOL is designed to be used in a similar way to an interpreted language like Prolog, Lisp or BASIC. To this end, the CIGOL interpreter resembles that of Prolog in appearance to the user, although the top-level prompt (!-) is different.

[1] For clarity, in this example we use an "illegal/3" predicate, where each argument is a two-place function symbol, rather than the "illegal/6" predicate used elsewhere in this work.

!- *illegal(wk(c,4),wr(h,1),bk(d,5)).*

. . .

I know:

illegal(wk(c,4),wr(h,1),bk(d,5)).

!- *illegal(wk(c,5),wr(h,1),bk(d,6)).*

. . .

TRUNCATION

Is illegal(wk(c,A),wr(h,1),bk(d,B)) always true ? *n.*

. . .

INTRA-CONSTRUCTION

illegal(wk(c,A),wr(h,1),bk(d,B)):-p39(A,B).
p39(4,5).
p39(5,6).

What shall I call p39 ? *adjacent_ranks.*

. . .

I know:

adjacent_ranks(4,5).
adjacent_ranks(5,6).
illegal(wk(c,A),wr(h,1),bk(d,B)):-adjacent_ranks(A,B).
not(illegal(wk(c,A),wr(h,1),bk(d,B))).

!-

Figure 15: CIGOL induces sub-concept "adjacent_ranks" in KRK.

Firstly, CIGOL is presented with an example of a position which is illegal because the two kings are diagonally adjacent on the board. A position is represented by a three-place predicate "illegal", or its negation. Note that this is the inverse of trying to learn "legal" as in Section 3.2. This aspect of formulating the top-level learning problem will be discussed further in Chapter 6. The three arguments of the illegal predicate are, in order, the squares occupied by the White King, White Rook and Black King in the position. Each square is denoted by its file and rank, as in the standard chess algebraic notation, so that the argument "wk(c, 4)" corresponds to "WK:c4".

When an example is presented in response to the interpreter prompt CIGOL immediately commences searching for potentially good applications of its inductive transformation operators. A good operator application is defined as one which enables program compaction. The ellipsis "..." in the figure indicates where messages from CIGOL regarding the progression of the search have been omitted for clarity. However, no generalisation by inverse resolution is possible from this first example. After each search stage has been completed, CIGOL shows the user all clauses currently in the program which is under construction prefaced with the header "I know: ".

Next, the user supplies a second example of a position which is illegal due to king adjacency. CIGOL again commences searching for potentially good applications of its inductive transformation operators. If one can be found, a good operator application is presented to the user as a query which includes the name of the applied operator, e.g. TRUNCATION, and the hypothesised transformation.

At this point, CIGOL finds a generalisation of the first two examples, and shows this to the user, requesting confirmation of its correctness *in all the cases which it covers*. However, the user rejects this suggestion, since it is not the case that all KRK positions where the White King is on file "c" and the Black King is on file "d" while the White Rook is on square "h1" are illegal.

Following this rejection, CIGOL re-commences the search, and subsequently

finds an application of the INTRA-CONSTRUCTION operator which is presented to the user. We can see from the figure that CIGOL has begun to "get the idea" with this second question, which concerns the adjacency of the files of the White and Black Kings in both of the examples provided. This, it will be remembered, is part of the reason why the positions are illegal, since in both the Kings are diagonally adjacent.

The application of the INTRA-CONSTRUCTION operator has created a new predicate, to which CIGOL has applied its own name, "p39". Here the user is asked not to confirm the correctness of a generalisation, but either to apply a meaningful name to the newly created predicate or to reject it as not useful. In this case the new predicate is useful as it is a partial definition of the concept of "adjacent_ranks", the name given by the user.

Following the successful application of the INTRA-CONSTRUCTION operator, the "adjacent_ranks" predicate name is incorporated into the set of clauses in the current program, which is again displayed on the terminal by CIGOL under the heading "I know: ". Note that the negative reply given to the suggested TRUNCATION above resulted in the addition to the current set of clauses of the *negation* of that suggested generalisation. This is because CIGOL now knows that **illegal(wk(c,A),wr(h,1),bk(d,B))** is *not* always true.

The session could have been terminated at this point, although other options are open to the user. These include adding further examples, or verifying the partial program developed so far by CIGOL against a test set of randomly chosen KRK positions. However, the example of Figure 15 illustrates the way in which the invention by machine of a pre-specified target program from raw data (tutorially chosen) may be carried out.

3.3.2 Correctness and comprehensibility

The native mode of the Duce and CIGOL systems is interactive. This exemplifies an incremental learning approach, where examples are added in stepwise

fashion to build the target program. As we have seen, both systems require an oracle. Owing to the fact that all induced concepts are adjudicated by the oracle, correctness and comprehensibility of the resultant program are ensured. Since the oracle is correct by definition, representing as it does a complete specification for the target program, no incorrect program clauses will ever be added. Therefore, if the initial program is correct, then the target program will be too. Comprehensibility of the resultant program is aided by the fact that any new terms hypothesised by CIGOL are given names by the oracle which are relevant to the task domain. This can be seen in Figure 15 where the induced two-place predicate is named "adjacent_ranks", which is meaningful for chess positions. For instance, the same relation in some other domain might have been named "successor".

Our investigations into the total automation of the concept-formation process in the KRK domain depend on running CIGOL, Duce and other systems for comparison, in non-interactive mode with large example sets. To this end non-interactive versions of both CIGOL and Duce have been developed in which the oracle has been automated. In effect, this means that every suggested operator application is automatically accepted. This usually leads to incorrect generalisations being incorporated into the induced programs. However, the effect of this incorrectness can be measured against test sets of examples. This performance metric provides one empirical basis for comparing implemented versions of inverse resolution with other candidate Machine Learning algorithms. We now discuss the experimental framework for these studies.

3.4 Experimental framework

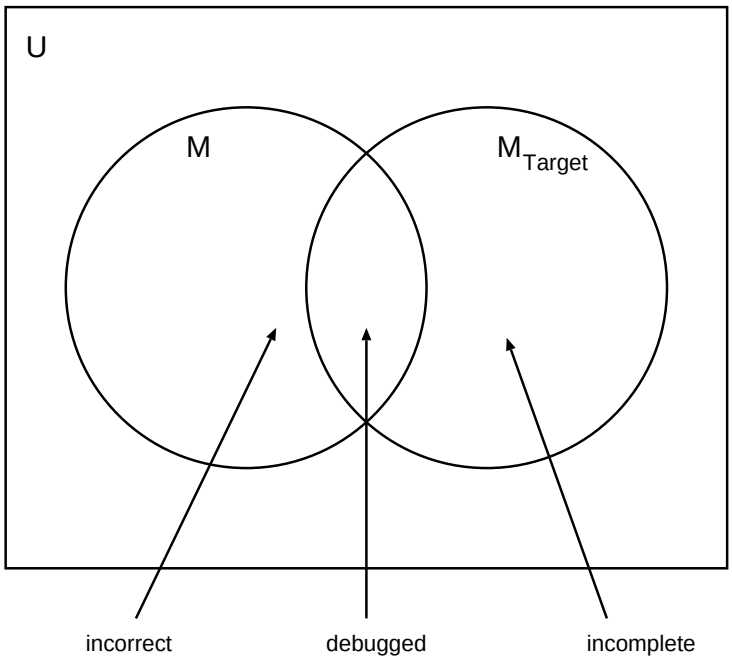
3.4.1 Incremental learning

We adopt a formulation of incremental learning based on that proposed by E. Shapiro [126] within which Machine Learning is viewed as automatic programming. In this framework the incremental learning of a target program is a process in which a sequence of programs is generated. Each of these may be incorrect or incomplete in the intended interpretation. If we take Logic Programming as our theoretical framework, we can refer to the intended interpretation as a model for the target program. It is important to note that in the sense in which we will define incorrectness and incompleteness of programs, a complete and correct program may be identified in the limit by an incremental learning algorithm [126]. Any terms used but not defined in the following discussion can be found in Robinson [115], Chang and Lee [18], or Lloyd [56, 57].

The Venn diagram of Figure 16 shows the relation between a current program P and some target program P_{Target}. The Least Herbrand Model of P is the set M. The Least Herbrand Model of P_{Target} is the set M_{Target}. M and M_{Target} are sets of ground literals. When the two sets M and M_{Target} are merged, i.e. when $M = M_{\text{Target}}$, then P is said to be equivalent to P_{Target}, and incremental learning terminates.

At any time, if $M \setminus M_{\text{Target}} \neq \emptyset$ then P is *incorrect* otherwise P is *correct*. Similarly, if $M_{\text{Target}} \setminus M \neq \emptyset$ then P is *incomplete* otherwise P is *complete*. The *debugged* part of the current program is represented in Figure 16 by $M \cap M_{\text{Target}}$. Correctness is more formally defined in Chapter 5.

If logic programs are considered as sets of clauses, then in the sequence of programs $P_0, P_1, P_2, \ldots$ generated by an incremental learning system, for $i < j < k$, if P_i contains a clause C found to be incorrect, and P_j does not contain C, then no P_k will contain C. This definition leads to the following general schema for an incremental learning system.



U	universe of ground literals
M	Least Herbrand Model of current program
M_{Target}	Least Herbrand Model of target program

Figure 16: Incremental learning - relationship of current and target programs.

For a logic program P :

repeat

if P is incorrect **then** *specialise* P

if P is incomplete **then** *generalise* P

until

P is complete and correct.

In Section 5.4.1 of Chapter 5 we define methods of specialisation and generalisation in the context of this schema. The incremental learning algorithms described in Chapters 6 and 7 are instances of this schema.

3.4.2 Hypothesis formation and testing

The schema for incremental learning given in the previous section is a generate-and-test strategy. Each step in the incremental learning process therefore comprises generation, or hypothesis formation, and testing. In the algorithms described in Chapters 6 and 7 the test step identifies exceptions (if any) to the current hypothesis. These are used in the next iteration of the incremental process. However, in the the comparative tests of Chapter 4 only the initial hypothesis formation and testing is considered, i.e. the algorithms are not incremental in the above sense.

A number of diverse strategies for hypothesis formation are employed by the different algorithms to be included in the comparative study of Chapter 4. Those of Duce and CIGOL are interactive, as illustrated in Sections 3.2 and 3.3. That of ID3 or its derivatives [107] is non-interactive, or offline, in inducing decision trees or rules. Therefore, to fit the experimental framework, hypothesis formation in Duce and CIGOL was constrained to operate in non-interactive mode, i.e. with all oracle queries answered positively.

The hypothesis formation sub-step becomes uniform for all algorithms and

consists of a training run requiring a single set of training examples. The hypothesis testing sub-step consists of a testing run requiring a single set of testing examples. The output of this testing stage gives the misclassification rate of the hypothesis on the test set. Therefore our performance criterion is test sample estimation of classification rate on the complete example population, as measured by mean error on a number of test sets.

Whether the induced hypothesis is formed by one of the "logic learning algorithms" (i.e. Duce or CIGOL), or some other (e.g. ID3), it may be considered as a classifier in the terms of Breiman et al. [11]. Assume that the prediction Y of a classifier must be in the set *Classes*. Denote the possible outcomes of classification during testing by Y or $\bar{Y}$. The latter means for a distinguished outcome Y, $Y \in$ *Classes* and $\bar{Y} \in$ *Classes* $- Y$. Then the definitions of incorrectness and incompleteness are:

Incorrect

Classifier predicts Y but test example labelled $\bar{Y}$

Incomplete

Classifier predicts $\bar{Y}$ but test example labelled Y

Note that for the logic learners the Closed World Assumption (CWA) is applied, i.e. if the attempt to prove Y finitely fails then conclude $\bar{Y}$.

3.4.3 Design of experiments

Dependent variable

Our primary aim is to investigate learning performance, which we will measure in terms of predictive accuracy. As discussed above, we will give results in terms of test sample estimation of the absolute misclassification rate.

Independent variables

In the comparative tests described in Chapter 4 two factors will be investigated to gauge their effect on learning performance.

Firstly formalism, or expressive power of the hypothesis language, is considered in designing experiments. This choice was constrained to include only algorithms with either a Zeroth Order Logic (ZOL) or First Order Logic (FOL) formalism or their equivalents.

Secondly sample size, as mentioned in Section 3.2, is varied. To avoid a large number of experimental conditions, the choice of size for training and test samples was restricted to being "small" or "large". The size difference should be of an order of magnitude. The choice of actual sizes for samples was ultimately dependent on the limitations of the most restricted algorithm (CIGOL as implemented in Prolog). However, since sampling was random and the domain size was much greater than the upper limit on sample size for CIGOL, it was decided that CIGOL should only be included in the experiments using small samples. Accordingly, the size of these samples was set to 100 examples for the small sample and 1000 examples for the larger sample.

Details of the experimental conditions applied in Chapters 6 and 7 are given where appropriate in each chapter.

3.5 Summary

We have given in this chapter further background and motivation for experimental studies on the induction of attributes from primitives. Preliminary work with two algorithms which learn in different formalisms was presented to show how the induction of attributes from primitives could proceed in the domain of KRK illegality. The chapter concluded with an overview of the general experimental framework as developed from the methods used in our exploratory trials.

Chapter 4

Formalism comparison

In this chapter we describe the results of a set of experiments in which we compared the learning performance of human and machine learning agents. The problem involved the learning of a concept description for deciding on the legality of positions within the chess endgame King and Rook against King. Various amounts of background knowledge were made available to each learning agent. We concluded that the ability to produce high performance in this domain was almost entirely dependent on the ability to express first-order predicate relationships.

4.1 Introduction

It is a commonly held belief that the use of a restricted hypothesis language simplifies the task of learning. In this chapter we investigate a simple problem in which this is not the case. We describe a set of experiments in which a number of different inductive learning agents, with various hypothesis languages, were provided with the same training and test material. In all the experiments described the training and test instances were selected from an instance space of size 262,144 using a standard random generator. Unlike many other comparative studies in the machine learning literature (e.g. [19, 120]) we have tested both human and machine learning agents. The machine learning algorithms involved

were Quinlan's C4 [108], Clark and Niblett's CN2 [19], Bratko et al's Assistant86 [16], Muggleton's Duce [80] and Muggleton and Buntine's CIGOL [86]. Although CIGOL is capable of constructing new predicates, this feature did not come into play in these experiment. Instead CIGOL's performance excelled in some experiments due to its use of a first-order hypothesis language. In all of these experiments the normally interactive algorithms CIGOL and Duce were run in automatic mode. In this mode oracle questions are automatically taken as being answered positively.

The learning problem involved deciding on the legality of positions in the chess endgame King-and-Rook against King. Despite the fact that predecessors of several of the machine learning algorithms involved were successfully developed and tested using chess endgame problems [106, 80], these same algorithms performed poorly in some of our experiments. The previous successes reported in the literature relied on the provision of special-purpose attributes which encoded relevant features of the board. Typically the use of such attributes dramatically reduces the size of the example space. In one of the experiments described in this chapter the only attributes provided were the coordinate values of pieces. This is the lowest level representation of positions used widely by chess players. In this experiment only CIGOL, a first-order learning system, required a small number of training instances to produce reasonable performance on the test set. This shows that the use of a restricted, propositional hypothesis language can prevent concepts from being learned efficiently.

This result confounds a common belief, which could be stated as follows. If a learning algorithm fails to produce predictive performance P within resource-bound R it would also fail to do so with a hypothesis space increased from H to H' ($H' \supseteq H$), since to find a high performance hypothesis you would have to consider at least H. To show the error of this argument consider Figure 17.
The tape diagram represents a linear ordering over two hypothesis spaces H and H' where $H' \supset H$. The common ordering represents a simplified version of what is often called learning *bias* [133]. Most practical learning algorithms use

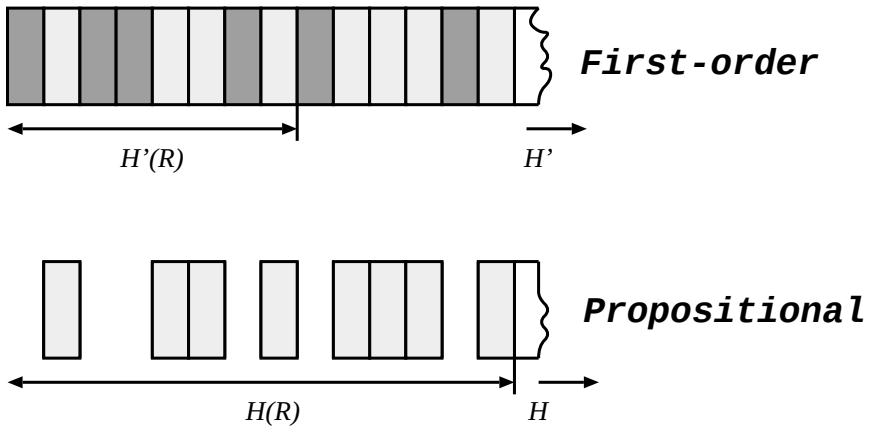


Figure 17: Hypothesis ordering, $r = 8$.

a simplicity criterion for this bias. A resource-bounded learning algorithm will only construct the first r hypotheses, limited by its resource bound R. The hypotheses in the diagram are of two different types, which might be thought of as first-order (striped) and propositional (dotted). Whereas a first-order algorithm will construct the r hypotheses $H'(R)$, a propositional algorithm with the same bias would construct the r hypotheses $H(R)$. This contradicts the belief that the first-order algorithm would have to consider at least the hypotheses considered by a propositional learning algorithm since $H'(R) \not\supseteq H(R)$. In the case exemplified by Experiment 1a (section 4.2.1), whereas there is a simple first-order description of part of the concept discoverable within the computational resource bounds, there is no corresponding simple propositional description.

4.2 Experiments

Four experiments were carried out.

1. Learning from piece-on-place attributes.

   (a) Small number of training instances (100). Involved CIGOL, Duce, C4, CN2, Assistant, humans.

(b) Large number of training instances (1000). Involved Duce, C4, CN2, Assistant.

2. Learning with extended hypothesis vocabulary

   (a) Small number of training examples (100). Involved CIGOL, Duce, C4, Assistant, humans.

   (b) Large number of training examples (1000). Involved Duce, C4, Assistant.

Owing to the diversity of the learning agents involved it was not possible to make the learning conditions identical for each agent. For instance, although it is possible to define and control the background knowledge available to machine learning algorithms, this can only be approximated in humans. Our subjects knew nothing of chess, but inbuilt spatial intuition is bound to have helped their ability to extract relations such as "collinear" from the example data.

4.2.1 Method

Experiment 1a - Without extended hypothesis vocabulary, small training set

Each machine learning algorithm was trained on five randomly generated sets of 100 instances. Each instance consisted of an illegal/legal class value paired with 6 attribute values. Each attribute value represented respectively the rank and file of each piece. An instance taken from one of the CIGOL training sets is shown in Figure 18.

The five rule sets induced by each machine learning algorithm except CN2 were tested on each of five randomly generated sets of 1000 instances, making 25 tests in total. Each of the five rule sets learned by CN2 was tested on only one set of 1000 examples.

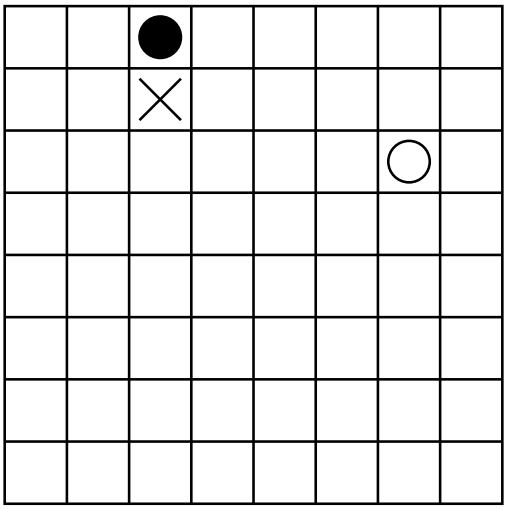
The human subjects consisted of 6 schoolchildren, aged 15-17, 3 boys and 3 girls. These subjects were provided with symbolic descriptions of instances

CIGOL (1a): illegal(g,6,c,7,c,8).

Human (1a):

						Yes	No
t	r	h	u	h	x	✓	

Human (2a):



Yes	✓	No	

Figure 18: Three representations of the illegal position WK:g6, WR:c7, BK:c8; White to move.
CIGOL(1a)'s representation is close to standard chess notation. In the representation of Human(1a) positions were coded using two separate substitutions for the "a,b,..,h" and "1,2,..,8" alphabets. The substitutions were respectively: "b,e,h,k,n,q,t,w" and "c,f,i,l,o,r,u,x". This encoding obscures while not removing the ordering of the alphabets. In the Human(2a) grid the symbols for White King, White Rook and Black King are ○, × and ● respectively.

similar to those provided for other learning systems, one of which is shown as Human(1a) in Figure 18.

One of the five randomly generated sets of 100 instances used in the machine learning experiments was selected as the training set for the human learning experiments. Similarly, one of the five randomly generated sets of 1000 instances used in the machine learning experiments was selected as the test set for the human learning experiments. However, to avoid errors due to fatigue human subjects were tested on only the first hundred of this test set. The experimental method for human subjects was as follows.

The experiment took the form of a multiple-choice questionnaire conducted under examination conditions. This was supervised by the author and the subjects' form teacher. Subjects were given a single paper in three sections. The first section was a tutorial introduction in which the task of generalising a predictive rule from example cases was presented using the game of noughts-and-crosses (tic-tac-toe). The second section contained the training examples and the third section contained the test examples. The KRK illegality task was disguised as the fictional computer game called "Ant Attack" (described in Section 1.1 of Chapter 1). In the second section of the paper all examples were labelled "yes" (illegal) or "no" (legal), as shown in Figure 18, Human(1a). The examples in the third section of the paper were not labelled. Subjects were instructed to tick either "yes" or "no" for each example in this section using the rules they had formulated by studying the examples in the previous section. Otherwise they should guess. All examples were required to be labelled. Subjects were allowed to refer between the different sections of the paper. At the end of the third section of the paper subjects were requested to write down in English the rules they had devised (if any) for classifying the unlabelled examples. Subjects were given five minutes to study the introductory section and then given one hour to complete the paper. Each paper was scored by counting the number of examples in the third section which had been correctly labelled. The individual and mean scores are shown in the Humans (1a) column of Figure 21.

Agents	Expt. 1a	Expt. 1b	Expt. 2a	Expt. 2b
Humans	51.2%, 1*hr*	N/A	79.3%, 1*hr*	N/A
CIGOL	84.2%, 1.5*hr*	N/A	77.2%, 21.5*hr*	N/A
C4	67.0%, 2.5*hr*	83.4%, 12.2*hr*	61.9%* 1.6*hr*	99.0%* 10*hr*
CN2	69.5%$^+$ 0.4*hr*	87.6%$^+$ 4*hr*	N/A	N/A
Assistant	55.7%, 0.25*hr*	56.2%*, 0.5*hr*	71.0%*, 0.25*hr*	91.0%*, 0.5*hr*
Duce	42.4%, 8*hr*	47.8%*, 10*hr*	33.7%*, 2*hr*	37.7%*, 10*hr*

Figure 19: Averaged final performance for each agent in each experiment together with approximate mean elapsed time for training and testing.
A '*' appears beside those values for which testing is not complete. A 'N/A' appears in those in which we are not attempting to carry out testing. A '+' indicates a variant training and testing regime, explained below.

Experiment 1b - Without extended hypothesis vocabulary, large training set

Each machine learning algorithm was trained on five randomly generated sets of 1000 instances. The test regime for all algorithms except CN2 was the same as that of Experiment 1a. Each of the five rule sets learned by CN2 was tested on one set of 100 instances.

Experiment 2a - With extended hypothesis vocabulary, small training set

Each machine learning algorithm was trained on the same randomly generated sets of 100 instances used for training in Experiment 1a, but in some cases extra background knowledge was supplied and in others the examples were presented in an extended hypothesis vocabulary. This was done as follows.

1. CIGOL and Duce - background knowledge predicate definitions for equality, adjacency and less than for files and ranks.

2. C4 and Assistant - hypothesis vocabulary extension consisting of all pairwise arithmetic differences between the integer file and rank values of all pieces.

These algorithms were tested on examples based on the same test instances as those used in Experiment 1a, using the same testing regime.

The testing of human subjects in Experiment 2a was conducted concurrently with that of Experiment 1a. The human subjects consisted of 7 schoolchildren, aged 15-17, 4 boys and 3 girls. The experimental method was identical to that of Experiment 1a, using the same randomly generated sets of examples, except for the format of the examples. Each example was presented as a diagram of an 8×8 array with circles and crosses in place of the pieces, shown as Human (2a) in Figure 18. Each diagram was marked "yes" (illegal) or "no" (legal). At no time was it suggested that the concept being learned concerned chess. The individual and mean scores are shown in the Humans (2a) column of Figure 21.

Experiment 2b - With extended hypothesis vocabulary, large training set

Each learning agent was supplied with the background knowledge used in Experiment 2a. The sets of training and test instances were the same as those used in Experiment 1b.

4.2.2 Results

In order to record the incremental performance change, we tested the performance against the entire test set in increments of 10 training instances. The Experiment 1a incremental performance figures for the all the machine learning algorithms except CN2, averaged over the 25 test runs, are graphed in Figure 20. A summary of the averaged final performance figures for each agent in each experiment together with the approximate mean elapsed time for training and testing is provided in Figure 19. A breakdown of the performance of individual human subjects together with the significance levels of their performances is provided in Figure 21.

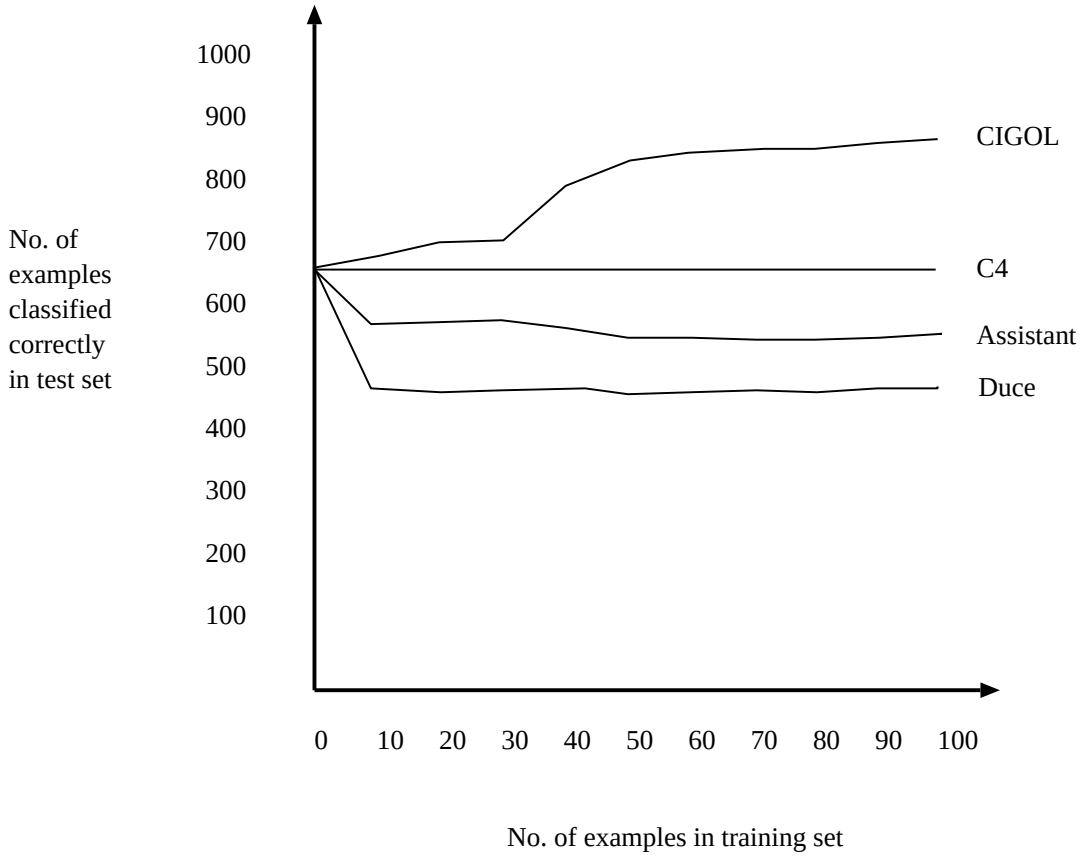


Figure 20: Incremental performance for Experiment 1a.

	Humans(1a)	Humans(2a)
Subject performance	$71^{***}, 71^{***}, 54, 47, 44, 20^{***}$	$98^{***}, 96^{***}, 92^{***}, 88^{***}, 64^{*}, 64^{*}, 53$
Group mean performance	51.2	79.3

Figure 21: Breakdown of human results.
Subject performances were tested for significance using 2×2 χ^2 test with Yates' correction. A '***' indicates a significance value of $p < 0.001$. A '*' indicates a significance value of $0.05 < p < 0.01$. Unmarked subject performance values are not significant at the 0.05 level.

4.3 Discussion

4.3.1 Experiment 1

CIGOL

The incremental performance graphs shown in Figure 20 are in many ways more informative than the final values shown in Figure 19. In experiment 1a all machine learning techniques start from a value of 67%. Since 67% of the instances in the instance space are legal, this is the "null" performance which would be expected from any system which assumes the default "everything is legal". Within the space of around 50 training examples CIGOL's performance rises to an average value of around 85% (91.4% maximum). In doing so CIGOL's hypothesis in Prolog is as follows.

```
illegal(A,B,C,D,A,B).  % The position is illegal iff the White King and
                       % the Black King are on the same square or
illegal(A,B,C,D,C,E).  % the White Rook and the Black King are on
                       % the same file or
illegal(A,B,C,D,E,D).  % the White Rook and the Black King are on
                       % the same rank.
```

Within this domain it is possible to analyse how many examples would be necessary for CIGOL to learn any particular unit clause. CIGOL needs at least two examples to be able to form an hypothesis such as "illegal(A,B,C,D,C,E)" by using its *truncation* rule. This would require two instances in which the White Rook and the Black King were on the same file. Imagining that we placed the White Rook on an arbitrary position on a chess board and then placed the Black King on another randomly chosen position the probability that they would lie on the same file is clearly $\frac{1}{8}$. However, CIGOL will often need more than two examples to make the generalisation "illegal(A,B,C,D,C,E)" since there is a $\frac{6}{8}$ chance that any two arbitrarily chosen instances of this rule will have a

corresponding rank or file value. This would lead to an hypothesis such as "illegal(A,3,C,D,C,E)", i.e. an under-generalisation. On the basis of this argument we would expect to require between $2 \times 8 = 16$ and $3 \times 8 = 24$ positive examples to develop the rule "illegal(A,B,C,D,C,E)". Since only one in three instances are "illegal" we would expect to require between 48 and 72 randomly chosen examples to develop the two *collinearity* rules "illegal(A,B,C,D,C,E)" and "illegal(A,B,C,D,E,D)". In practice this takes between 40 and 80 examples, much as predicted. The fact that such analysis is possible points to an advantage of carrying out this kind of experimentation within a closed and analytically tractable domain.

It is also easy to see that CIGOL's performance will not ever rise to 100%. The reason is that the collinearity rules, although allowing rapid promotion to high performance, are overgeneralisations. Exceptions exist to these rules when the White King is interposed between the White Rook and Black King. Since CIGOL learns monotonically, it is not possible to correct such overgeneralisations. Specialisation techniques to overcome this problem are the focus of the remaining chapters of this thesis.

CN2 and C4

CN2 and C4 produced very similar performance, with performance almost indistinguishable from the performance of the null rule "every position is legal" in Experiment 1a. Both performances rise gradually when presented with ten times as much training data to a more reasonable 88% and 83% respectively.

Assistant and Duce

The performance of both Assistant and Duce rapidly diminishes from an initial 67%. In Assistant's case performance levels out at 56%, whereas Duce levels out at 48% (Expt 1b).

These poor performances by C4, Assistant and Duce can be partly explained by the fact that a complete description of collinearity within a decision tree propositional formalism is very large and thus needs a large number of examples to justify such an hypothesis. With some work it might be possible to predict just how many examples would be required in the same way as we have done for CIGOL.

Humans

At first sight the human mean performance value given in Figure 19 looks close to that produced by random guessing. However this is clearly not the case when we look at the individual scores in Figure 21. As evidenced by the starred significance indications, individual scores are strongly polarised into those that found an effective prediction method and those which merely guessed. In all cases but one insignificant performances agree with the children's reporting that they merely guessed on the answer sheet. The exception to this is the unexplained 20% score which produces a highly significant score with "YES" and "NO" reversed.

4.3.2 Experiment 2

CIGOL and Duce

Information on the time taken for each learning agent to reach the performance levels shown in Figure 19 helps to understand these results. CIGOL typically takes longer to learn by an order of magnitude when supplied with background knowledge than in Experiment 1a. Usually none of the background predicates appear in the final hypothesis so they do not add to predictive power. However the number of predicates in the knowledge base is doubled when the background knowledge is included. This enlarged search space means that CIGOL is unable to find the best hypotheses within available resources.

Duce and CIGOL are machine learning algorithms which construct their

own background predicates when doing so simplifies the problem. It was this capability that suggested that they might be appropriate candidates for this learning task. However, we now realise that the achievement of high performance within this domain is *not* dependent on the availability or constructibility of background predicates, but rather a problem of having a sufficiently expressive formalism. This seems to contradict the results of strong performance of C4 and Assistant given appropriate background knowledge. However, a glance at the form of necessary background knowledge for C4 and Assistant's strong performance (Section 4.2.1) shows that it is essentially relational, i.e. could only be expressed in a First-order language. The usually vague notion of "background knowledge" in this case conceals a change of formalism.

Humans

The question of formalism also appears in the human results. When asked to describe the rules that they were applying all successful candidates gave rules similar to the following.

Concept is true

If black nought is in the same line as the cross

If white nought is right next to the black nought

If white and black noughts are in the same box

If black nought is in the same box as the cross

The use of relational concepts in this natural language description is evident.

4.4 Summary

From the results provided in this chapter, our main conclusion is that the ability to produce high performance in this domain is almost entirely dependent on the ability to express first-order predicate relationships. This ability may

be conferred either by a first-order logic hypothesis language or via relational attributes encoded in a propositional logic.

Chapter 5

Non-Monotonic Learning

This chapter addresses methods of specialising first-order theories within the context of incremental learning systems. We demonstrate the shortcomings of existing first-order incremental learning systems with regard to their specialisation mechanisms. We prove that these shortcomings are fundamental to the use of classical logic. In particular, minimal "correcting" specialisations are not always obtainable within this framework. We propose instead the adoption of a specialisation scheme based on an existing non-monotonic logic formalism used in logic programming. We define the minimal correcting specialisation of a normal logic program and present an algorithm for its construction. This approach overcomes the problems that arise with incremental learning systems which employ classical logic.

5.1 Introduction

5.1.1 Motivation

Generalisation is not everything in learning. New experience can often require the specialisation of over-general beliefs. For example imagine that you believed

that all birds fly. We might write this as

$$Flies(x) \leftarrow Bird(x)^{1} \tag{1}$$

If you were told that although an emu is a bird it cannot fly you would have to specialise (1) to deal with this exception. In this chapter we discuss various ways in which this could be done. In the following we will assume that an incremental learning algorithm, as described in Section 3.4.1 of Chapter 3, receives new examples one at a time, its belief set being revised after each example. The belief set is generalised when it does not cover a new example. On the other hand it is specialised when contradicted by a new example. Most work in machine learning is based on the related notions of generalisation and specialisation. However, most treatment of the topic of specialisation has been within the context of non-incremental learning algorithms such as ID3 [106], AQ11 [61], INDUCE [63] and Version Space algorithms [75]. There are exceptions to this, such as E. Shapiro's debugging system and some recent work on theory revision systems. These are discussed below in Section 5.1.2 and Section 5.4.2. In this chapter we investigate the topic of incremental specialisation. Clearly an incremental learning algorithm as described above changes the coverage of its beliefs *non-monotonically*, hence the title of this chapter. However, in Section 5.1.3 we show that the developers of logic-based machine learning algorithms to date have avoided non-monotonic logic representations. This leads to various problems. In Section 5.2 we prove that it is not possible in general to preserve correct information when incrementally specialising within a classical logic framework. In Section 5.3 we demonstrate that this impasse can be easily avoided by learning algorithms which employ a non-monotonic knowledge representation.

The CIGOL algorithm [86] is not incremental in the sense of the description above. Although it generalises from positive examples it does not specialise when it encounters negative examples which contradict its theory. In practice this has led to difficulties. As reported in Chapter 4, by over-generalising CIGOL

[1] The logic notation used in this chapter is defined in Appendix A.

managed to outstrip the performances of both humans and propositional learning algorithms when incrementally learning the KRK illegality concept from randomly selected examples. However, having over-generalised and reached a performance level of around 90%, CIGOL was not able to produce 100% performance since it could not specialise the concept definition. In this chapter we lay the theoretical foundations for incremental specialisation techniques. Implementations and testing of these techniques are reported in Chapters 6 and 7.

5.1.2 Generalisation and specialisation

Niblett [90] has shown that the concept of generality can be expressed within the framework of logic. Let P and Q be two well-formed-formulae. P is more general than Q if and only if $P \vdash Q$. We might also say equivalently that P is a generalisation of Q or Q is a specialisation of P. Note that since P and Q can be any well-formed-formulae they might be atomic formulae, clauses or conjunctions of clauses.

5.1.3 Previous incremental specialisation techniques

E. Shapiro [126] describes an incremental Program Debugging System (PDS) which recognises three different types of bug within Horn clause logic programs. Given a Horn clause program P, a ground unit goal G and the intended interpretation M (set of ground atoms) of P, P is said to be *incomplete* when $G \in M$ and $P \nvdash G$, *incorrect* when $G \notin M$ and $P \vdash G$ and *non-terminating* when $G \in M$ but G leads to a non-terminating SLD-resolution proof from P. For the purposes of this chapter we are interested in the case in which PDS finds P to be *incorrect*. In the general case, when PDS finds P to be incorrect with respect to G it searches for a clause C which covers G and removes C from P. Thus let P' be the resulting program where $P = P' \wedge C$. Note that clause removal is a specialisation technique since $P' \wedge C \vdash P'$. However removing C from P is a somewhat drastic

method of specialising a logic program since P' may now become incomplete with respect to a goal G' in M previously covered by C. In the "bird" example of Section 5.1.1 let M be a superset of $\{Flies(Eagle), Bird(Eagle), Bird(Emu)\}$, $P = \{(Flies(x) \leftarrow Bird(x)), (Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow)\}$, $G = Flies(Emu)$ and $G \notin M$. As $G \notin M$ and $P \vdash G$, PDS would delete the clause $Flies(x) \leftarrow Bird(x)$ leaving $P' = \{(Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow)\}$. Since P' is incomplete with respect to $G' = Flies(Eagle)$, PDS would generalise P' to $P'' = \{(Flies(Eagle) \leftarrow), (Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow)\}$. However, whereas the goal *Flies(Sparrow)* could be proved from $P \cup \{Bird(Sparrow)\}$ it cannot be proved from $P'' \cup \{Bird(Sparrow)\}$. In this chapter we will be investigating less drastic specialisation techniques than clause removal.

Wrobel [141] describes a program called MODELER which incrementally learns theories in a clausal logic formalism without function symbols. MODELER's knowledge revision module applies a heuristic approach to deal with Shapiro's incorrectness problem. MODELER saves exceptions to each rule in the form of a support set. Having found a sufficiently large set of exceptions MODELER introduces a new unary predicate to describe the exceptions. The clauses are then reformulated in terms of the new predicate. The approach used in MODELER has some similarities to that described in Section 5.3 of this chapter. However since MODELER, like PDS, does not use a non-monotonic logic representation it also tends to over-specialise when presented with counter-examples. In the bird example, with P, M and G the same as above MODELER would produce $P'' = \{(Flies(x) \leftarrow Bird(x) \wedge Concept1(x)), (Concept1(Eagle) \leftarrow), (\neg Concept1(Emu) \leftarrow), (Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow)\}$. Again, whereas *Flies(Sparrow)* could be proved from $P \cup \{Bird(Sparrow)\}$ it cannot be proved from $P'' \cup \{Bird(Sparrow)\}$.

The approaches taken in these two machine learning programs are rather typical of attempts to date to design algorithms which produce incremental specialisation. However, if either of these techniques were applied to the theory

of KRK illegality produced by CIGOL (discussed in Section 5.1.1) the performance would drop drastically from a strong performance of around 90% correct to a much weaker performance of around 67% correct. The reason is that both PDS and MODELER over-specialise. But how could one avoid over-specialising P? In Section 5.2 we define and investigate the properties of the most-general-specialisation P' of a first-order clausal theory P such that $P \vdash G$ and $P' \nvdash G$. Although at first this seems a promising approach, we demonstrate that P' can contain an indefinitely large set of clauses. In Section 5.3 we show that this problem can be avoided by the using a non-monotonic formalism. As an introduction to non-monotonic formalisms in Section 5.1.4 we briefly describe some of the background. This approach leads to new problems in defining the notions of generality and redundancy within this formalism.

5.1.4 Non-monotonic formalisms

One of the most common approaches to non-monotonic reasoning is based on the "Closed World Assumption" (CWA) inference rule. According to the CWA if a ground atom A is not a logical consequence of a theory then infer $\overline{A}$ [56]. The CWA can be implemented in two different ways. First, by adding additional completion axioms to the theory and applying standard theorem proving techniques. This approach is exemplified by "predicate completion" [56] and "circumscription" [28]. Second, by employing "Negation by Failure" (NF) where a modified theorem prover infers $\overline{A}$ whenever the attempt to prove a ground atom A finitely fails. The second approach is used within the logic programming language Prolog for which NF has been shown to be equivalent to "predicate completion" [56].

5.2 Most-general-correct-specialisation (mgcs)

In this section we define the most-general-correct-specialisation (mgcs) of an incorrect clausal theory and prove various theorems leading to a resolution-based method for constructing an mgcs. The lengthy proof of Theorem 5.7 from this section has been placed in Appendix B to improve readability. In the course of proving Theorem 5.7 we defined a function called "deriv" which turned out to be an improvement on the Kedar-Cabelli and McCarty [42] explanation-based-generalisation algorithm. Prolog code and a description of the relationship between "deriv" and the algorithm described in [42] are also given in Appendix B. First we define the correctness of first-order formulae in a weaker way than Shapiro's definition (Section 5.1.3). Throughout the following definitions and theorems we use the term "intended interpretation" to mean the abstract model of an unknown formula. In any implementation we would only expect to know the truth of some ground atoms from the intended interpretation of a formula.

Definition 5.1 (Correctness) *Let F be a well-formed first-order formula and M be the intended interpretation of F. We say that F is correct with respect to M, or simply F is correct, whenever M is a model of F. F is said to be incorrect otherwise.*

An implementation of this definition would allow for correction of a formula with respect to known facts.

Lemma 5.2 (Correct-conjunction) *Let F be the conjunction of well-formed formulae $(F_1 \wedge .. \wedge F_n)$ and M be the intended interpretation of F. F is correct with respect to M if and only if each F_i is correct with respect to M.*

Proof. *Follows trivially from the fact that M is a model of $(F_1 \wedge .. \wedge F_n)$ if and only if M is a model of each F_i.*

Definition 5.3 (Correct-specialisation) *Let T and T' be sets of first-order clauses and M be the intended interpretation of T. T' is said to be a correct-specialisation of T if and only if T' is correct with respect to M and $T \vdash T'$.*

Definition 5.4 (Mgcs) *Let T and T' be sets of first-order clauses and M be the intended interpretation of T such that T is incorrect with respect to M and T' is a correct-specialisation of T. The set of clauses T' is said to be the most-general-correct-specialisation of T if and only if $T' \vdash T''$ for every clause set T'' which is a correct-specialisation of T.*

This definition assumes the existence of a unique mgcs for every clausal theory. The proof of Theorem 5.11 provides a resolution-based method for constructing such an mgcs, which guarantees its existence. Before stating this theorem we need to prove some intermediate results.

Lemma 5.5 (Subsumption by refutation) *Let A and B be two well-formed first-order formulae. $A \vdash B$ if and only if $A \wedge \overline{B} \vdash \square$.*

Proof. $(A \vdash B) \equiv (A \vdash B \leftarrow \blacksquare) \equiv (A \vdash \square \leftarrow \overline{B}) \equiv (A \wedge \overline{B} \vdash \square)$ *by the Deduction Theorem.* $\square$

In the following we assume familiarity with Robinson's [115] results on resolution theorem proving. Robinson defines $R^n(T)$, for a set of clauses T, as follows

$$\begin{aligned} R^0(T) &= T \\ R^n(T) &= R^{n-1}(T) \cup \{C : C_1, C_2 \in R^{n-1}(T), \\ &\qquad C \text{ is the resolvent of } C_1 \text{ and } C_2\} \end{aligned}$$

In addition we will define the *resolution closure* of a set of clauses as follows.

Definition 5.6 (Resolution closure) *Let T be a set of clauses. The resolution closure of T, $R^*(T)$ is $(R^0(T) \cup R^1(T) \ldots)$.*

The resolution closure of T does not contain all of the clauses entailed by T. The following theorem describes the relationship between the clauses entailed by T and the clauses within the resolution closure of T.

Theorem 5.7 (Clause entailment using resolution) *Let T be a set of clauses and C be a non-tautological clause. $T \vdash C$ if and only if there exists D in $R^*(T)$*

and substitution θ such that $D\theta \subseteq C$.

Proof. *See Appendix B.*

Lemma 5.8 (Theory entailment) *For any two sets of clauses T and T', $T \vdash T'$ if and only if $T \vdash C$ for each clause C in T'.*

Proof. *Let T' be represented by the conjunction $(C_1 \wedge .. \wedge C_n)$. According to Lemma 5.5, $(T \vdash T') \equiv (T \wedge \overline{T'} \vdash \Box) \equiv (T \wedge \overline{(C_1 \wedge .. \wedge C_n)} \vdash \Box) \equiv T \wedge (\overline{C_1} \vee .. \vee \overline{C_n} \vdash \Box) \equiv ((T \wedge \overline{C_1}) \vee .. \vee (T \wedge \overline{C_n}) \vdash \Box)$ (2). But $((T \wedge \overline{C_i}) \vdash \Box) \equiv (T \vdash C_i)$ by Lemma 5.5. Thus (2) is true if and only if $T \vdash C$ for each clause C in T'.* $\Box$

Lemma 5.9 (Theory entailment using resolution) *Let T and T' be two sets of clauses. $T \vdash T'$ if and only if for every clause C in T' there exists a clause D in $R^*(T)$ and a substitution θ such that $D\theta \subseteq C$.*

Proof. *Follows trivially from Lemma 5.8 and Theorem 5.7.*

The following definition is similar to Plotkin's definition of θ-subsumption [100] and is used in the statement of the main theorem, Theorem 5.11.

Definition 5.10 (θ-subsumption) *We say that clause A θ-subsumes clause B or $A \sqsupseteq B$ if and only if there exists a substitution θ such that $A\theta \subseteq B$. Similarly, we write $A \sqsupset B$ when $A \sqsupseteq B$ and $B \not\sqsupseteq A$.*

Theorem 5.11 (Resolution-based mgcs construction) *Let T be a set of clauses and M be the intended interpretation of T. Let $Q = \{\ C : C \in R^*(T)$ and C correct wrt $M\ \}$ be the correct part of $R^*(T)$ and $S = \{E : E$ is a clause which is correct wrt M and there exists $D \in R^*(T) - Q$ and $D \sqsupseteq E$ and for every E', $D \sqsupseteq E' \sqsupset E$ only if E' incorrect wrt $M\}$ be a specialisation of the incorrect part of $R^*(T)$. $Q \cup S$ is the mgcs of T.*

Proof. *From Definition 5.4, $Q \cup S$ is the mgcs of T if and only if $Q \cup S$ is a correct-specialisation of T and $Q \cup S \vdash Q'$ for every clause set Q' which is a correct-specialisation of T. From Definition 5.3, $Q \cup S$ is a correct-specialisation of T if and only if $Q \cup S$ is correct with respect to M and $T \vdash Q \cup S$. Using*

Lemma 5.2, $Q \cup S$ is correct with respect to M by construction since each clause in $Q \cup S$ is correct with respect to M. Also, using Theorem 5.8, $T \vdash Q \cup S$ since T entails each clause in $Q \cup S$. Thus $Q \cup S$ is a correct-specialisation of T.

We must now prove that $Q \cup S \vdash Q'$ for every clause set Q' which is a correct-specialisation of T. Assume that there exists Q' which is a correct-specialisation of T and $Q \cup S \nvdash Q'$. Thus, applying Lemma 5.9, it is not the case that for every clause F, F is in Q' only if there exists a clause G such that G is in $R^*(Q \cup S)$ and $G \sqsupseteq F$. That is to say there exists a clause F such that F is in Q' and for every clause G, G is in $R^*(Q \cup S)$ only if $G \not\sqsupseteq F$. But since Q' is a correct specialisation of T, $T \vdash Q'$. Thus, by applying Lemma 5.9 either there is a clause H in Q such that $H \sqsupseteq F$ or there is a clause I in $(R^*(T) - Q)$ such that $I \sqsupseteq F$. However, since $Q \subseteq R^*(Q \cup S)$ the first alternative contradicts the assumption letting $G = H$. Therefore we must assume the second alternative, that $I \sqsupseteq F$. However, from the definition of S it can easily be shown that for every correct clause J for which there is a clause D in $R^*(T) - Q$ there is a clause E in S such that $E \sqsupseteq J$. Thus either F is incorrect with respect to M, which it is not since Q' is correct wrt M, or there is an E in S and $E \sqsupseteq F$. Letting $G = E$ this contradicts the assumption and completes the proof. □

Theorem 5.11 would seem to provide the basis for an algorithm which could enumerate the elements of the mgcs of an incorrect theory T. However, this does lead to some difficulties, as the following example shows.

Example 5.12 *We continue the "bird" example of Section 5.1.3. Let T be the set of clauses $\{(Flies(x) \leftarrow Bird(x)), (Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow)\}$. Let the true ground atoms in the intended interpretation M of T be a superset of $\{Flies(Eagle), Bird(Eagle), Bird(Emu)\}$, where $Flies(Emu)$ is not true in M. T is incorrect with respect to M since the clause $(Flies(x) \leftarrow Bird(x))$ is incorrect for $x = Emu$. Constructing the sets in Theorem 5.11 we get $Q = \{(Bird(Eagle) \leftarrow), (Bird(Emu) \leftarrow), (Flies(Eagle) \leftarrow)\}$ and $S \supseteq \{(Flies(Emu) \leftarrow \neg Bird(Emu)), (Flies(x) \leftarrow Bird(x) \wedge \neg Bird(Emu))\}$. Imagine that $Flightless(x)$*

is true in M for all those x which cannot fly and false for all those which can fly then the clause $(Flies(x) \leftarrow Bird(x) \wedge \neg Flightless(x))$ is an element of S. But if we assume that M may contain the predicate "Flightless" then, since no restriction is placed on the vocabulary (alphabet) of the theory, M could contain an indefinitely large number of predicates which could be used in combination to produce an indefinitely large number of additional clauses within S.

In the next section we show that by assuming the existence of additional predicates such as "Flightless" and by using a non-monotonic representation we can solve the problem of over-specialisation of incorrect theories.

5.3 Closed World Specialisation

In the previous section we showed that most general correct specialisation (mgcs) can lead to theories which are not finitely axiomatisable. In this section we introduce a method of avoiding this problem by making use of the "negation as failure" rule in the context of normal logic programs. Instead, we develop our method of Closed-World Specialisation using semantic definitions for normal logic programs. Following Lloyd [57], we begin by defining normal logic programs.

Definition 5.13 (Normal program clause) *A normal program clause is a clause of the form $A \leftarrow L_1, \ldots, L_n$ where A is an atom and $L_1, \ldots, L_n$ are literals.*

Definition 5.14 (Normal logic program) *A normal logic program is a finite set of normal program clauses.*

In the remainder of this section we will assume that all normal logic programs under discussion are *stratified*. Stratification is defined as follows, based on the definitions given in [57].

Definition 5.15 (Level mapping) *A level mapping of a normal logic program P is a mapping from the set of predicate symbols in P to the non-negative integers. The value of a predicate symbol under this mapping is the level of that predicate symbol.*

Definition 5.16 (Stratification) *A normal logic program is stratified if it has a level mapping such that in every program clause $p(t_1, \ldots, t_n) \leftarrow L_1, \ldots, L_m$, the level of the predicate symbol of every positive literal in the body is less than or equal to the level of p, and the level of every negative literal in the body is less than the level of p.*

We note in section 5.3.2 that our Closed-World Specialisation algorithm does not violate this assumption.

As we stated in Section 5.1.4 the logic programming language Prolog uses negation as failure. In classical logic the ground literal $\overline{A}$ is provable from theory T if and only if $T \vdash \overline{A}$. In Prolog the ground goal *not(A)* is provable from program P if and only if A is an atom and $P \nvdash A$. The usual proof method for Prolog is SLDNF resolution ("linear resolution with selection function using the negation as failure rule"). We now give definitions for the semantics of normal logic programs under SLDNF resolution.

There has been a great deal of discussion recently on the semantics of general logic programs, a class which includes normal logic programs, e.g. [136, 58, 104, 41]. For simplicity we will assume that the model of a normal logic program P is equivalent to the success set of P under SLDNF resolution. The set of all ground atoms which can be constructed from predicates and function symbols in P is the Herbrand base of P, $H(P)$. The success set of P is the set of ground atoms $A \in H(P)$ such that there is a (finite) SLDNF-refutation for $P \cup \{\leftarrow A\}$. We use the notation $Ms(P)$ to denote the set of models of a program P. $Ms(P)$ is a subset of the power set of the Herbrand base of P, $Hs(P)$. We now prove a basic theorem in the semantics of normal logic programs.

Theorem 5.17 (Unique model for a normal logic program) *For a normal logic program P, $M(P)$ is unique.*

Proof. *Assume the opposite, that there is a normal logic program P with more than one model, call them M1 and M2. Let M1 contain a ground atom a entailed by P which is not in M2. But a is entailed by P, and so M2 cannot be a model for P, which contradicts the assumption. This proves that there is at most one model for a normal logic program. A program P' containing the empty clause has no models, but the empty clause is not a normal program clause by Definition 5.13, so P' is not a normal logic program by Definition 5.14.* □

In Section 5.2 we defined the generality relation between sets of clauses in terms of entailment. Then $P \models P'$ iff $\mathrm{Ms}(P) \subseteq \mathrm{Ms}(P')$. If as above, $\mathrm{Ms}(P)$ and $\mathrm{Ms}(P')$ are singletons (unit sets), then this condition only applies if $P = \square$, which is not a normal logic program, or $P' \models P$, i.e. $P \equiv P'$. Consequently we cannot use entailment, as in the previous section, to define the generality lattice and mgcs. In the remainder of this section, we will restrict our attention to the semantics of normal logic programs, and write $M(P)$ to denote the unique model for such a program P. This model is a subset of $H(P)$.

Following Lloyd [57] we refer to the *definition* of a particular predicate symbol p in a normal logic program P. The definition is taken to be the set of all program clauses in P which have p in their head. We consider the semantics for the definition of a predicate in normal logic programs. This is because our algorithm may introduce a new predicate into the first order language for specialised programs, thus complicating the semantical relation between the program and its specialisation. We use $M(p|P)$ to denote the model for a particular predicate p in a program P. This model is a subset of the Herbrand base for p in P which we write as $H(p|P)$.

Definition 5.18 (Predicate model) $M(p|P) = \{a : a \in M(P), pred(a) = p\}$, *where pred(a) denotes the predicate symbol of atom a.*

Since we cannot use entailment, we use the result of Theorem 5.17 on the unique model for a normal logic program to define a specialisation relation between programs.

Definition 5.19 (Generality by model inclusion) *Let P_1 and P_2 be normal logic programs. Then P_1 is more general than P_2 iff $M(P_1) \supseteq M(P_2)$. Alternatively we say that P_2 is a specialisation of P_1. Similarly, the definition of predicate symbol p in P_1 is more general than the definition of p in P_2 iff $M(p|P_1) \supseteq M(p|P_2)$.*

As in Section 5.2, we require the concept of "intended interpretation", to be thought of as the abstract model for some target program. In the context of inductive logic programming, a learning example is an element of the intended interpretation. More formally, we have the following definition.

Definition 5.20 (Learning examples) *Let $\oplus$ be a set of positive examples and $\ominus$ a set of negative examples, such that each $e \in \oplus \cup \ominus$ is a ground atom and $\oplus \cap \ominus = \emptyset$. Then for the intended interpretation $M(P)$ of a target normal logic program P, $e^+ \in \oplus$ only if $e^+ \in M(pred(e^+)|P)$ and $e^- \in \ominus$ only if $e^- \notin M(pred(e^-)|P)$.*

We proceed to define the most general correct specialisation of a predicate in a normal logic program with respect to a negative example. In the following we consider the definitions of a predicate symbol p in the normal logic programs $P, P', P'', \ldots$ with respect to a negative example $e^- \in \ominus$.

Definition 5.21 (Correctness of a predicate) *Let P be incorrect with respect to e^- iff $e^- \in M(pred(e^-)|P)$, and correct otherwise.*

Definition 5.22 (Correct specialisations of a predicate) *Let P' be a correct specialisation of P with respect to a negative example e^- iff P' is correct with respect to e^- and $M(pred(e^-)|P) \supseteq M(pred(e^-)|P')$. We shorten this to $P' \in CS(P, e^-)$.*

Definition 5.23 (Most general correct specialisations of a predicate) *Let P' be a most general correct specialisation of P with respect to a negative example e^- iff $P' \in CS(P, e^-)$ and for every $P'' \in CS(P, e^-)$ we have $M(pred(e^-)|P') \supseteq M(pred(e^-)|P'')$. We shorten this to $P' \in MGCS(P, e^-)$.*

In the next section we give an algorithm for computing the closed-world specialisation $P' = \mathrm{CWS}(P, e)$ of a predicate in a normal logic program with respect to a negative learning example and prove that $P' \in \mathrm{MGCS}(P, e)$.

5.3.1 Closed World Specialisation Algorithm

In this section we will represent clauses in notation based on Edinburgh Prolog [21]. For example the clause $(P(x) \leftarrow Q(x) \wedge R(x))$ is written in Prolog as

p(X) :- q(X), r(X).

But note that the Prolog clause

p(X) :- q(X), not(r(X)).

states that p(X) is provable if q(X) is provable and r(X) is not provable. Note that such a clause is a normal program clause (Definition 5.13). The literal to the left of the ":–" is called the "head" of the clause, while the set of literals to the right of the ":–" is called the body of the clause. Prolog proofs are carried out using SLDNF-resolution. The definitions of SLDNF refutation, finitely-failed SLDNF tree and computed answer substitution are given in Lloyd [57].

The algorithm for our Closed World Specialisation technique is presented in Figure 22. We now show that the closed-world specialisation $P' = \mathrm{CWS}(P, e^-)$ of a predicate in a normal logic program with respect to a negative learning example is a most general correct specialisation, $P' \in \mathrm{MGCS}(P, e^-)$. To do this we will use the following result.

Lemma 5.24 (Model of the most general correct specialisations of a predicate) *The set $MGCS(P, e^-) = \{P' : M(pred(e^-)|P') = M(pred(e^-)|P) -$*

CWS Algorithm

Input: Normal logic program P and ground atom e^- such that
$P \vdash e^-$ and $e^- \in \ominus$

For every normal clause $C \in P$ which can be resolved with e^-
in an SLDNF refutation of $P \cup \{\leftarrow e^-\}$

Let θ be the computed answer substitution from
the SLDNF refutation of $P \cup \{\leftarrow e^-\}$

If the body of C contains a literal *not(B)* then

Let $P' = P \cup \{B\theta \leftarrow\}$

Else

Let C be the clause $A \leftarrow (B_1, \ldots, B_m)$

Let $\{V_1, .., V_n\}$ be the variables in A

Let q be a predicate symbol not found in P

Let B be $q(V_1, .., V_n)$

Let $P' = P - \{C\} \cup \{A \leftarrow (not(B), B_1, \ldots, B_m)\} \cup \{B\theta\}$

Output: P'

Figure 22: Closed World Specialisation (CWS) Algorithm.

$\{e^-\}\}$.

Proof. *The result is obvious from Definition 5.23 since $M(pred(e^-)|P')$ must be the largest subset of $M(pred(e^-)|P)$ not containing e^- for any $P' \in MGCS(P, e^-)$.*

□

Theorem 5.25 (CWS Algorithm constructs element of MGCS) *Let $P' = CWS(P, e^-)$. Then $P' \in MGCS(P, e^-)$.*

Proof.

We refer to the CWS algorithm of Figure 22. For each normal program clause $C \in P$ as defined in the algorithm there are two cases to consider. First, the case where the body of C contains a literal $not(B)$. For each element e' apart from e^- in the model $M(pred(e^-)|P)$ either e' resolves with some clause other than C, call it $D \in P$, which does not affect the model $M(pred(e^-)|P')$ since $D \in P'$, or e' resolves with C in an SLDNF refutation of $P \cup \{\leftarrow e'\}$ with substitution θ'. Then for every $B\theta'$ there is a finitely-failed SLDNF tree for $P \cup \{\leftarrow B\theta'\}$. Let θ be the

computed answer substitution from the SLDNF refutation of $P \cup \{\leftarrow e^-\}$. *Clearly* $e^- \in M(pred(e^-)|P)$ *and there is a finitely-failed SLDNF tree for* $P \cup \{\leftarrow B\theta\}$. *In this case* $P' = P \cup \{B\theta\}$, *so there is an SLDNF refutation of* $P' \cup \{\leftarrow B\theta\}$ *and the goal* $not(B)\theta$ *fails. Therefore* $e^- \notin M(pred(e^-)|P'$. *Since the specialisation to* P' *adds no other* $B\theta'$ *apart from* $B\theta$ *to* P *there is a finitely-failed SLDNF tree for* $P' \cup \{\leftarrow B\theta'\}$ *for every* θ' *apart from* θ, *so* $M(pred(e^-)|P')$ *contains every* e' *apart from* e^-. *Therefore, using Lemma 5.24,* $P' \in MGCS(P, e^-)$.

Second, consider the case where the body of C *does not contain a literal* $not(B)$. *As in the first case for each element* e' *apart from* e^- *in the model* $M(pred(e^-)|P)$ *either* e' *resolves with some clause* $D \in P$ *other than* C, *which does not affect the model* $M(pred(e^-)|P')$ *since* $D \in P'$, *or* e' *resolves with* C *in an SLDNF refutation of* $P \cup \{\leftarrow e'\}$ *with substitution* θ'. *Then for every goal* $G = \leftarrow (B_1, \ldots B_m)\theta'$ *there is an SLDNF refutation of* $P \cup \{G\}$. *Let* θ *be the computed answer substitution from the SLDNF refutation of* $P \cup \{\leftarrow e^-\}$. *Then for the goal* $G = \leftarrow (B_1, \ldots B_m)\theta$ *there is an SLDNF refutation of* $P \cup \{G\}$, *so* $e^- \in M(pred(e^-)|P)$. *In this case* $P' = P - \{C\} \cup \{A \leftarrow (not(B), B_1, \ldots, B_m)\} \cup \{B\theta\}$, *there is an SLDNF refutation of* $P' \cup \{\leftarrow B\theta\}$ *and the goal* $not(B)\theta$ *fails. Therefore* $e^- \notin M(pred(e^-)|P'$. *As in the first case, since the specialisation to* P' *adds no other* $B\theta'$ *apart from* $B\theta$ *to* P *there is a finitely-failed SLDNF tree for* $P' \cup \{\leftarrow B\theta'\}$ *for every* θ' *apart from* θ. *Consequently* $M(pred(e^-)|P')$ *contains every* e' *apart from* e^- *and, using Lemma 5.24,* $P' \in MGCS(P, e^-)$. □

In the following we show how the CWS algorithm operates on the "bird" example.

Example 5.26 *Let* P *be the set of clauses* {*(flies(X) :- bird(X)), (bird(eagle) :-), (bird(emu) :-)*} *and the true ground atoms in* $M(P)$, *the intended interpretation of* P, *be a superset of* {*flies(eagle), bird(eagle), bird(emu)*}, *where* e^- = *flies(emu) is not in* $M(P)$. P *is incorrect with respect to* e^- *since the clause* C = *(flies(X):–bird(X)) in the definition of the predicate symbol flies is incorrect with substitution* $\theta = \{X/emu\}$. *The body of* C *does not contain a*

literal not(B). $\{X\}$ is the domain of θ. Let q be "flightless". B is flightless(X) and P' is $\{$(flies(X):- bird(X), not(flightless(X))), (bird(eagle):–), (bird(emu):–), (flightless(emu):–)$\}$. Now flies(emu) is not provable from P' so P' is correct with respect to e^-. Furthermore, unlike the methods in Section 5.1.3, flies(sparrow) can be proved from $P' \cup \{$(bird(sparrow) :–)$\}$.

Two implementations in Prolog of the Closed World Specialisation Algorithm have been completed. The first version of the algorithm forms part of the CIGOL system, and is discussed in Chapter 6. The second is a stand-alone version used with the GOLEM system. This is discussed in Chapters 6 and 7.

5.3.2 Stratification

We follow Definition 5.16 of stratification for normal logic programs. For a stratified normal logic program P, $P' = \mathrm{CWS}(P, e)$ is also stratified, since the clause head of the specialised predicate depends negatively on a literal which is understood to be assigned a level strictly lower than that of the clause head. Our method of specialisation when combined with generalisation implemented by CIGOL or GOLEM as presented in Chapters 6 and 7 also does not violate stratification for the sequence of programs $P_0, P_1, P_2, \ldots$ in an incremental learning session. This is discussed further in Chapter 7.

5.4 Discussion

The Closed World Specialisation algorithm transforms a normal logic program given a single negative example. However in practice it is often required that a program be corrected with respect to a set of negative examples. Furthermore this situation may occur many times within an incremental learning process which includes alternate training and testing phases. Such testing may identify also sets of positive examples which are not covered by the program in addition to negative examples which are covered by the program. We now discuss extending

the CWS algorithm to implement incremental learning from batches of examples. The incremental framework is based on that of Section 3.4.1 in Chapter 3.

5.4.1 Incremental learning

For a learning task we envisage a sequence of programs $P_0, P_1, P_2, \ldots$ generated by an incremental learning system. The intended interpretation for the target program P_{Target} is M_{Target}. This is a set of ground atoms from which learning examples in the sense of Definition 5.20 are sampled. Learning begins with a training set $T_0 \subseteq M_{\text{Target}}$ provided as input to the learning system which outputs program P_0. This is the initialising step of the incremental process. All subsequent steps consist of the following pattern. First, program P_{i-1} is tested with respect to a set of learning examples $\oplus \cup \ominus$ to produce a set of exceptions. The exceptions consist of the elements of $\ominus$ (negative examples) which are entailed by P_{i-1} and the elements of $\oplus$ (positive examples) which are not entailed by P_{i-1}. Second, the exceptions are used in the batch specialisation and generalisation of P_{i-1}. The processes of batch specialisation and generalisation, described below, transform clauses in P_{i-1} and produce a set of ground unit clauses. In the final step, these ground unit clauses are themselves generalised and added to P_{i-1} thus giving the new program P_i.

Note that the test set of learning examples at any step in the incremental process is expected to be disjoint from the training set. At each step a new training set is formed from the union of the previous training set and the exceptions set. If the exceptions set is empty then the incremental process is terminated and the current program is output. We proceed to describe batch specialisation and generalisation relative to a set of exceptions.

Consider the case of a normal logic program P which is incorrect with respect to all negative examples in the exceptions set. A batch specialisation P' of a normal logic program P with respect to a set of exceptions $E^- \subseteq \ominus$ may be constructed by calling the CWS algorithm on each element of E^-.

Definition 5.27 (Batch specialisation) *The batch specialisation $P' = BS(P, E^-)$ of a normal logic program P with respect to a set of negative examples E^- is returned by the function*

$$BS(P,\ E^-) = \begin{cases} P & \text{if } E^- = \emptyset \\ BS(P',\ E'^-) & \text{otherwise if } e^- \in E^- \text{ and } E'^- = E^- - e^- \\ & \text{and } P' = CWS(P,\ e^-) \\ \textit{undefined} & \text{otherwise} \end{cases}$$

It follows from Theorem 5.25 that each specialisation step in Definition 5.27 is minimally correcting since either $E^- = \emptyset$ in which case $P' = P$ or for each $e^- \in E^-$ the resulting program $P' = \mathrm{CWS}(P, e^-)$ is an element of $\mathrm{MGCS}(P, e^-)$.

A batch generalisation P' of a normal logic program P with respect to the set of all positive examples $E^+ \subseteq \oplus$ in the exceptions set is defined as follows.

Definition 5.28 (Batch generalisation) *The batch generalisation $P' = BG(P, E^+)$ of a normal logic program P with respect to the set of positive examples E^+ is the set $P \cup E^+$.*

Using these batch transformations within a single step of an incremental process in which a possibly incorrect and incomplete program P undergoes least specialisation and least generalisation will return the program P'. Then P' will be complete and correct with respect to the examples in the test set. This is an important property of our incremental method. In contrast, ID3 with windowing [106] has no equivalent stage where the hypothesis is minimally corrected and generalised with respect to exceptions in the current test set.

Following the steps of batch specialisation and generalisation the current theory contains ground atoms. For the case of specialisation these are instances of predicates other than the target (i.e. top-level) predicate. Batch generalisation results in the addition of ground atoms to the definition of the target predicate. In both cases these atoms may be generalised with respect to the theory and negative examples in the training and test sets. Generalising the

set of positive examples added by batch generalisation may produce an over-general, i.e. incorrect program with respect to the target model. However, such over-generalisations can obviously be corrected on the subsequent cycles of the incremental process. Generalising the set of instances of an exception predicate will specialise its parent clause. When such generalisations are made, it is important to ensure that consistency between the exception predicate and its parent is maintained. In our method this is accomplished by storing the set of all training examples used to induce the initial theory and all exception examples used to update subsequent versions of the theory. This prevents any generalisation which covers negative examples, and any specialisation which does not cover positive examples.

We note that in this incremental method no clause will contain more than one negated literal, i.e. the exception literal. This point also relates to the introduction of non-negated exception literals, which is described in Chapter 7. Additionally, we do not consider the incremental learning of programs where the target predicate is invoked by some other predicate. In such cases, changing the model of the target predicate by generalisation or specialisation affects the model of the invoking predicate. It may be possible to formalise this constraint on the invocation of the target predicate by defining the stratification restrictions required. Further discussion on using our methods for incremental learning is in Chapter 8.

5.4.2 Related work

The specialisation of an over-general theory as defined above may be viewed as a belief revision problem. Belief revision is an important problem in Artificial Intelligence. It has been widely studied within Knowledge Representation, Logic Programming, Probabilistic Reasoning and Non-monotonic reasoning, as well as Machine Learning. It also may be seen as part of disciplines outwith the scope of this thesis, like Neural Networks. We briefly discuss below some work on

specialisation, primarily in the context of Machine Learning.

As the name suggests, in the Closed-World Specialisation (CWS) algorithm, the Closed-World Assumption [111] is applied. The "exception" predicates constructed by the CWS algorithm under the Closed-World Assumption are related to McCarthy's use of "abnormality" predicates within the framework of circumscription [59]. There is also a relation to Vere's method of learning "counterfactuals" in classical logic [137]. This is discussed further on page 125. Within an incremental learning framework, the combination of CWS with methods of generalisation leads to a process of non-monotonic inductive inference as discussed in [39].

However, we began in Section 5.1.3 by identifying deficiencies in certain specialisation systems which could lead those systems to over-specialise. In developing our system we were motivated to avoid over-specialisation. To the extent that we have developed a specialisation system, we relied on an assumption common to such systems. This has been termed "the competent programmer assumption" by E. Shapiro [126] and others. On this view, at any stage in the incremental learning process the current program is probably largely correct. Testing of the program can confirm if this is justified. Therefore it is likely to be more efficient to modify the current program than jettison it completely and start again from scratch. This approach is shared by many Machine Learning systems, including those discussed in the remainder of this section. It is also a feature of work in automated program debugging which builds on Shapiro's system. Dershowitz and Lee [24] describe a debugging system for Pure Prolog which relies on the provision of executable specifications. The specifications, also in Pure Prolog, are assumed to be correct, and are executed to provide test cases which can serve to locate bugs. Debugging is implemented by a set of heuristic operators on Horn clauses, such as adding literals (for specialisation) and adding clauses in a method similar to abduction (for generalisation).

Although, as we stated in Section 5.1.4, non-monotonic logic is a well explored topic, the non-monotonic representations that have been developed to

date seem not to have been applied in machine learning. However, based on an earlier version of the work described in this chapter, Ling [54] has developed a related method relying on SLDNF resolution. It appears, though, that his method may not give minimal specialisations which are finitely axiomatisable. His proposal is to augment a specialisation method based on Shapiro's refinement operator [125]. Other work by Ling [53] using a system called SIM relies on heuristic-guided abstraction operators, which are inversions of refinement operators. Ling's motive is an identification in the limit result for SIM, which he obtains. However, it is clear that incorrect clauses can appear in the logic program being constructed by SIM at any point during the process of incremental learning. Any such clause is removed by Shapiro's contradiction backtracing algorithm on being found to prove a negative example.

The main difference between Closed-World Specialisation and a specialisation system based on refinement operators is that the latter methods are restricted to operation in a finite language of observational terms. Such a framework does not allow the addition of new theoretical terms to the representation. In a closed system, this may not be an unreasonable restriction. Unfortunately, in a growing representation, a refinement operator over the hypothesis language may generate indefinitely large sets of clauses in searching for a specialisation of an incorrect clause. A similar problem exists for our proposed resolution-based method for constructing an MGCS, as Example 5.12 shows. To cope with the default assumptions required in expressing a minimal specialisation we employ a non-monotonic inference method. After each stage of the incremental learning process a check may be made on the current program. This is to determine whether the current program meets the current specification, and all previous specifications. Then the user may terminate the incremental learning process at any stage and have confidence as to the behaviour of the current program. In this context, the specification is an example set which should be covered by a complete and correct program. Muggleton [77] has pointed out the correspondence between this incremental learning approach to program synthesis (in an

Inductive Logic Programming framework) and the deductive approach of formal methods as described by Hoare [37]. Under this reading, in both ILP and formal methods the fundamental relation is that the target program ought to imply its specification. The correspondence is a broad one, since in ILP, unlike formal methods, it is generally assumed that the specification may be incomplete.

Recently, some work in knowledge-base refinement or theory revision has had as a stated goal the minimal change to a set of clauses when updates are carried out. It appears however that learning in a growing language may be a problem for some of these systems. As we have mentioned, Ling's work suffers from reliance on an approach based on refinement operators. Also, although he has proposed using negation as failure (NF) and an equality theory to construct a most general specialisation of a set of clauses, this may not give a result which is finitely axiomatisable. Within a finite language, Ling and Valtorta [55] and Craw and Sleeman [22] have given results for the complexity of the refinements produced by their systems. However, in both cases the representation is restricted to propositional rules, although Ling and Valtorta have uncertainties attached to the rules. In their representation, which is similar to that of the expert system MYCIN [127], complexity results relate to the intractability of certain strength refinements where these uncertainties are updated. The latter method is based on generating multiple rule refinements and heuristically filtering those considered suitable. Hamakawa [33], working with a propositional Horn-clause representation, points out the problem of over-specialisation and the need for incremental learning. Ginsberg [29] provides a frequent reference point for work on theory revision. Again, his system uses a set of heuristic refinement operators and a propositional representation.

A number of theory revision systems which are related to our method include those using logic programs as a representation. EITHER [92, 93, 94] was restricted to a propositional representation, but was a precursor of the first-order system FORTE [113, 76]. This system is designed to implement generalisation as well as specialisation. The specialisation operators used in FORTE consist

of deleting a clause from the theory or adding a literal to a clause. Heuristics are used to guide a hill-climbing search strategy for the "optimal" revision. AUDREY, a similar system at least in the respect of the specialisation methods used, is described by Wogulis [140].

A slightly different approach is taken in systems which operationalize or partially evaluate the theory before revision. For example, in the recent system of Tankitvanich et al. [131], the faulty theory is operationalized using Explanation-Based Learning (EBL) techniques, faults are detected and then specialisation carried out using methods of first-order learning. This system does not appear to avoid possible over-specialisation. Other systems that carry out intermediate operationalization steps include FOCL [97] and ML-SMART [6].

Minimal specialisation according to our definitions is not reported for the above theory revision systems. A proposal to use criteria other than minimal specialisation for theory revision has recently been made by Wrobel [142]. This is a formalisation of the exception list method implemented in MODELER, discussed above in Section 5.1.3. The particular criteria he suggests are based on the set of Gärdenfors postulates [27] for belief revision. Unfortunately this framework depends on a classical logic representation in which belief sets are closed theories. In this representation revisions may not be finitely axiomatisable, which is similar to the problem illustrated by Example 5.12. Therefore, Wrobel employs a modified set of postulates without the requirement for closed theories. Instead a postulate is added requiring that the revised theory contain only clauses from the original theory together with clauses θ-subsumed by clauses in the original theory. This set is termed the "base revision postulates". Wrobel then shows that his exception list method meets the base revision postulates.

Owing to the requirement that revised clauses be θ-subsumed by incorrect clauses, this method is similar to other Machine Learning methods which delete clauses or add literals, as discussed above. Although the method is shown to be minimal with respect to the base revision postulates, it may construct infinitely long exception lists, i.e. the revised theory may not be finitely axiomatisable.

Wrobel suggests that exception lists may be replaced by invented predicates. However, this would result in a revised theory which no longer meets the base revision postulates.

In Logic Programming the problem may be expressed in terms of updating logic databases. For instance, Guessoum and Lloyd [31, 32] have recently presented results for provably correct updates on normalised logic programs. This is an interesting technique which takes advantage of the soundness of SLDNF resolution to insert or delete an atom from a logic program. Deletion and insertion are carried out by update procedures which return a set of database transactions, given the program, atom and SLDNF-tree. A transaction is a set of assert or retract actions to be carried out on the program.

The specialisation operation is the deletion of an atom from the program. Given a program P and an atom A the first step is to construct an SLDNF-tree T for $P \cup \{\leftarrow A\}$. To delete the atom A from P, the update procedure returns a set of database transactions which retract program clauses corresponding to input clauses on non-failed branches of T and assert unit clauses corresponding to negated literals which were called and succeeded on non-failed branches of T. Ensuring that there is at least one input clause to be retracted or one unit clause to be asserted for each non-failed branch of T means that A is no longer derivable from P. This is in some respects similar to the proof of Theorem 5.25, although their method does not guarantee minimal specialisation when an atom is deleted from a program.

As these and other authors have noted, the similarities between aspects of logic program debugging, logic database updates, abduction, induction and non-monotonic reasoning are tantalising, although formalisation of the various relationships is still in progress. The relation of such work with Machine Learning is discussed further in Chapter 8.

5.5 Summary

From our practical experience gained with the CIGOL learning system in Chapter 4 it was found necessary to have a specialisation mechanism in order to produce monotonic performance increase. In this chapter we have shown that the "minimal specialisations" necessary to achieve this monotonic performance increase cannot be achieved in a classical logic representation (Section 5.2). They can however be achieved using a non-monotonic representation (Section 5.3). We believe that this is the first time that a minimally specialising algorithm has been demonstrated for normal logic programs. In Chapters 6 and 7 we describe the implementation and testing of the algorithm.

In this treatment of error-correction within logic programs we have not addressed the issue of noise within our Closed World Specialisation scheme. Clearly it is important not to be too hasty in constructing new exception predicates when dealing with noisy data. Although in this thesis the learning data may be assumed to be error-free, our method has elsewhere been augmented with a noise-detection mechanism based on a compression metric using the algorithmic complexity model of induction [129, 43, 17, 79]. The results so far from experimental testing of this approach are promising [79].

We have shown in this chapter the improvements over Shapiro's specialisation method gained by our approach, but we have not shown how our CWS algorithm may be properly integrated into an incremental learning framework such as that of PDS. In the following chapters we will describe this in the context of two different generalisation methods and give results from applying the combined systems to significant tasks in chess endgame domains.

Chapter 6

Experiments in non-monotonic learning

The technique of Closed-World Specialisation was developed in Chapter 5 to address the problem of correcting first-order theories within a non-monotonic framework for incremental learning. In this chapter we report on experiments combining this technique with two methods of generalisation in first-order logic. A new inductively generated solution giving 100% predictive accuracy is presented for the KRK illegality task.

6.1 Introduction

This chapter summarises the results of two experiments in non-monotonic learning. In both experiments, the generalisation methods employed were those of learning in a Horn clause subset of first-order logic. To be precise, in the first experiment a version of Muggleton and Buntine's system CIGOL [86] was used, and in the second experiment this was replaced with Muggleton and Feng's GOLEM system [87]. Both generalisation methods were combined with the Closed-World Specialisation (CWS) technique described in Chapter 5 for "correcting" over-general logic programs.

The experimental task in both cases was the induction of a logic program to recognise illegal chess positions where only the white king (WK), the white rook (WR) and the black king (BK) are on the board (the KRK endgame) with white's side to move (WTM). Randomly placing the three pieces on the chess board squares gives $64^3 = 262144$ possible configurations. Of these approximately one third are illegal. If two or more pieces are placed on the same square, or if the kings are placed on two vertically, diagonally or horizontally adjacent squares then the position is illegal. In addition, since we assume WTM, if the black king is in check then the position is illegal. This type of position forms the largest sub-class of the KRK illegality concept.

Two example KRK positions are illustrated in Figure 23. The position shown on the left is illegal since BK:e1 is in check with WTM. The position shown on the right is legal, i.e. not illegal. The six arguments of the *illegal* predicate give file and rank coordinates for, in order, the WK, WR and BK. Each of the three pieces takes two arguments, for file and rank values, from respectively {a, ..., h} and {1, ..., 8}. Note that this Prolog representation of positions is based on the standard algebraic notation used for representing human chess games [138].

Despite the simplicity of KRK [65], the task of inducing classification rules to distinguish legal from illegal positions highlights the shortcomings of induction in non-relational formalisms, as evidenced by the results of Chapters 3 and 4. This is of practical importance, for instance, in the area of induction over databases, where typically the formalisms are relational. Consequently the KRK illegality induction task has become something of a benchmark in relational learning studies (for example [85, 109, 50]).

6.2 Experiment 1: NM-CIGOL

This experiment followed from the work reported in Chapter 4 in which several algorithms, including CIGOL, were compared on the task of learning the concept

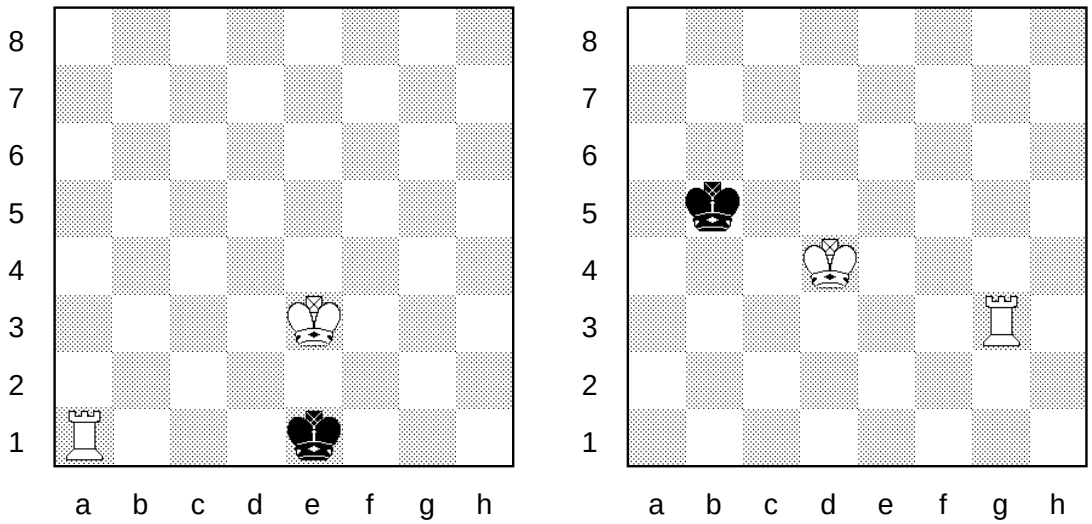


Figure 23: Two example KRK positions with their Prolog representations. On the left, the position is illegal since BK:e1 is in check with WTM. On the right, the position is legal, i.e. not illegal. The 6 arguments of the illegal predicate give file and rank coordinates for, in order, the WK, WR and BK. Each of the three pieces takes two arguments, for file and rank values, from respectively {a, ..., h} and {1, ..., 8}.

of KRK illegality. NM-CIGOL is CIGOL plus the CWS algorithm presented in Chapter 5. To illustrate the operation of NM-CIGOL compared with CIGOL we give below two partial traces of their interactive usage. The instance and hypothesis language is a subset of "Edinburgh" Prolog [9]. Both systems were implemented in Prolog and in the following execution traces were invoked from within the Prolog interpreter. User input is shown in a bold font and system output in a typewriter font.

Firstly, consider the case in which CIGOL is given the hypothesis '**p(A)**'. (This could equally well be an induced hypothesis). CIGOL is then given the exception '**not(p(a))**'. However, this input causes an error, and is ignored.

```
| ?- cigol.

!- p(A).

  ⋮

I know:
     p(A).

!- not(p(a)).

** ERROR: p(a) is always true

I know:
     p(A).

!-
```

Next we see how NM-CIGOL incorporates such exceptions by specialising the hypothesis. The following illustrates how the system processed the above case.

```
| ?- cigol.

!- p(A).

  ⋮
```

```
I know:
        p(A).

!- not(p(a)).
  ⋮

I know:
        p(A) :- not(nonp1(A)).
        nonp1(a).
        not(p(a)).

!-
```

As described in Chapter 5 the CWS algorithm invents a new predicate to cover the exception. In the above example the new predicate was given the machine-generated predicate symbol "nonp1".

6.2.1 Method

The goal of the experiment was to test whether NM-CIGOL's solution performed better than the previous most accurate solution learned by CIGOL for KRK illegality. NM-CIGOL was run in batch mode, meaning that all oracle (user) questions were taken as being answered positively. In the following note that "theory" is used as a synonym for logic program, and $P \vdash G$ will be taken to mean "$P \wedge \bar{G}$ is refutable using SLDNF resolution" (see [56]). The initial theory in this experiment was the best-performing theory induced by CIGOL from 100 examples in Experiment 1a, Chapter 4. This theory was found to be 91.4% correct on the test set, as discussed in Section 4.3.1 of Chapter 4. All example positions were randomly generated then classified by database lookup. The experimental method fits the framework described in Section 3.4 of Chapter 3 and Section 5.4.1 of Chapter 5. The incremental method was as follows:

0. Initialise :

T_0 = theory induced by CIGOL from 100 examples

Training set $R = 5000$ examples

Test set $S = 5000$ examples

1. Train :

   $\imath = 0$

   do

   $\imath = \imath + 1$

   Form exceptions set $E_\imath$

   if $E_\imath = \emptyset$ **then break**

   $T_\imath = \text{NM-CIGOL}(T_{\imath-1}, E_\imath)$

   until resources exhausted.

2. Test :

   $\jmath = 0$

   do

   $\jmath = \jmath + 1$

   $\text{Pscore}_\jmath = |\{l : l \in S, T_\jmath \vdash l\}|$

   $\text{Nscore}_\jmath = |\{l : l \in S, T_\jmath \vdash \bar{l}\}|$

   $\text{Result}_\jmath = \frac{\text{Pscore}_\jmath}{\text{Pscore}_\jmath + \text{Nscore}_\jmath}$

   until $\jmath = \imath$.

The formation of the exceptions set $E_\imath$ uses *theory-guided sampling* from the (very large) training set. A function **rand_subset(Size, Set)** was used to return a random subset of exceptions, sampling with replacement. The size of exceptions set was limited to $0 \leq |E_\imath| \leq 20$ due to resource constraints. The method was as follows, where a model M for the target theory is a set of ground literals.

Theory $T_{\imath-1}$ partitions training set :

$P = \{p | p \in M, T_{i-1} \vdash \bar{p}\}$

$N = \{n | \bar{n} \in M, T_{i-1} \vdash n\}$

if $|P| > 10$ **and** $|N| > 10$ **then**

$E_i = rand\_subset(10,\ P) \cup rand\_subset(10,\ N)$

else if $|P| > 10$ **and** $|N| \leq 10$ **then**

$E_i = rand\_subset(20 - |N|,\ P) \cup N$

else if $|P| \leq 10$ **and** $|N| > 10$ **then**

$E_i = P \cup rand\_subset(20 - |P|,\ N)$

else $E_i = P \cup N$.

The **rand_subset** function therefore implements a form of theory-guided sampling [1]. This is related to Quinlan's technique of windowing as used in ID3 and its derivatives [106], and stratified sampling [14].

6.2.2 Results

The working hypothesis was that NM-CIGOL would improve on the initial theory. Recall that this initial theory was found to give the maximum performance in Experiment 1a of Chapter 4, recording 91.4% predictive accuracy. Predictive accuracy is expressed as the percentage $\left(\frac{a+c}{a+b+c+d}\right) \times 100$ calculated from the following 2×2 table:

		Machine Guesses		
		"illegal"	"legal"	
Actual	illegal	a	b	a+b
Values	legal	d	c	c+d
		a+d	b+c	a+b+c+d

[1] Use of this technique was suggested to the author by Professor D. Michie

However, in the current experiment the performances of this theory (Cycle 0) were slightly lower due to the use of different example sets. This would be the base level from which the predictive accuracy was hypothesised to increase. The results were as follows.

Cycle	Classification performance (a)	(b)	CPU (seconds) (c)
0	90.14	89.26	124.2
1	90.54	89.26	3921.3
2	90.94	89.26	4439.3
3	91.22	89.20	9131.4
4	85.12	83.46	57277.0
4*	92.38	90.24	N/A

Cycle 4*: results without

illegal(A,B,C,B,D,E):-not(nonillegali4j1(A,B,C,D,E)).

(see section 6.2.3 below).

In the second, the predicate "succ/2" (14 clauses) was supplied as background.

Cycle	Classification performance (a)	(b)	CPU (seconds) (c)
0	90.14	89.26	126.4
1	90.54	89.26	4688.2
2	90.22	88.46	18116.7
3	84.06	81.38	40806.7
4	84.08	80.94	73800.0*

* Estimated time, process terminated on Cycle 4.

Key to measures :

(a) Predictive accuracy (%) on training set (5000).

(b) Predictive accuracy (%) on test set (5000).

(c) User + system, for training (except Cycle 0) plus testing;
(Quintus Prolog Release 2.5.1, Sun Sparcstation 330).

6.2.3 Discussion

The results are the opposite of the expected improvement on the initial theory, showing a drop in predictive accuracy on training and test sets. This does however mask performance gains which were evident when considering the effect of *specialisation* and *theory-guided sampling* on certain clauses in the induced theories.

Note the tendency in the preceding tables for performance to decline over successive cycles. This was caused by a combination of over-generalisation of certain clauses due to limited numbers of examples, and under-generalisation of the corresponding exception clauses. Typically, on a given cycle instances of illegality due to king adjacency were over-generalised by CIGOL. These generalisations were under-constrained since there were insufficient examples in the training sets. Exceptions to these over-generalisations were then incorporated by specialisation, carried out by CWS. However, the instances of new exception predicates invented during this specialisation were often not generalised by CIGOL. Therefore the "parent" clause remained over-general.

On subsequent cycles the same process was repeated, with the consequence of adding more and more incorrect clauses to the theory on each cycle. The performance therefore decreased over a number of cycles. In effect, the incremental learning system was diverging rather than converging on the target concept.

6.3 Experiment 2: CW-GOLEM

The negative result in experiment 1 was mainly due to insufficient examples of king adjacency in the training sets. In an attempt to rectify this, a second

experiment was planned with many more examples in the training set at each stage in the incremental procedure. Improvements in resultant performance were expected through the use of Closed World Specialisation (CWS) coupled with efficient generalisation using GOLEM. Although a non-monotonic version of GOLEM is planned, for the purposes of this work all generalisation and specialisation steps were treated as separate sub-routines of a combined system named CW-GOLEM. The implementation of GOLEM was in C. The version of CWS used was implemented in Quintus Prolog.

6.3.1 Method

In outline, this experiment consisted of three main steps.

1. **Generalisation:** Using GOLEM, a theory was induced from a training set of 1000 examples.

2. **Testing:** The induced theory was tested on a separate test set of 10000 examples. Those misclassified were saved as exceptions.

3. **Exception handling:** In two stages:

    a. **Specialisation:** A new version of the CWS algorithm, extended to generate negative examples of the exception predicates, was used to correct the induced theory.

    b. **Generalisation:** The positive and negative examples of the exception predicates were generalised with respect to background knowledge using GOLEM.

These steps roughly correspond to a single cycle of an incremental procedure similar to that given in Section 6.2.1. The following example is provided to illustrate the combination of theory-guided sampling of exceptions, specialisation and generalisation in correcting a clause which partially describes the "black king in check" rule.

Example.

Let T_i be the logic program for KRK illegality resulting from the application of GOLEM to 1000 randomly selected pre-classified examples (step 1). Let M^+, M^- be subsets of the target model M. Specialisation must be preceded by a test phase where each ground literal in the test set $M^+ \cup M^-$ is called as a goal on T_i (step 2). Let $C_i \in T_i$ be the "culprit" clause to be specialised. $P \subseteq M^+$ is the set of ground literals covered by C_i and no other clause in T_i. $N \subseteq M^-$ is the set of ground literals such that $\bar{n} \in N$ and n covered by C_i and no other clause in T_i. Thus each goal in the set $P \cup N$ may be said to be "covered alone" by $C_i \in T_i$.

Below we give a particular case for goals $r, s \in P$ and $t, u \in N$. V is the set of variables in C_i, and L is a new predicate symbol not in the vocabulary of T_i. Following specialisation we have new clause C_{i+1}, positive literals $L\theta_t$ and $L\theta_u$, and negative literals $L\theta_r$ and $L\theta_s$ (step 3a). With respect to background predicates including "lt/2" ("<" for chess-board coordinates), X is the the set of hypothesised clauses produced by GOLEM to cover the positive and none of the negative exception literals (step 3b). The new theory T_{i+1} results from T_i by replacement of the culprit clause C_i with C_{i+1} and addition of the generalised exceptions clause X.

C_i = illegal(A,B,C,D,C,E).

$P \ni r$ = illegal(a,2,a,6,a,4).
$P \ni s$ = illegal(b,2,a,6,a,4).

$N \ni t$ = not(illegal(a,4,a,6,a,2)).
$N \ni u$ = not(illegal(b,3,b,7,b,1)).

$V = \{A, B, C, D, E\}$
L = "between_file/5"

$\theta_r = \{A/\mathrm{a}, B/2, C/\mathrm{a}, D/6, E/4\}$

$L\theta_r$ = between_file(a,2,a,6,4).

$\theta_s = \{A/\text{b}, B/2, C/\text{a}, D/6, E/4\}$

$L\theta_s$ = between_file(b,2,a,6,4).

$\theta_t = \{A/\text{a}, B/4, C/\text{a}, D/6, E/2\}$

$L\theta_t$ = between_file(a,4,a,6,2).

$\theta_u = \{A/\text{b}, B/3, C/\text{b}, D/7, E/1\}$

$L\theta_u$ = between_file(b,3,b,7,1).

X = GOLEM(between_file(a,4,a,6,2),
between_file(b,3,b,7,1), ...,
not(between_file(a,2,a,6,4)),
not(between_file(b,2,a,6,4), ...,
lt(0,1),lt(0,2),lt(0,3), ...)

= between_file(A,B,A,C,D) :- lt(D,B), lt(B,C).

C_{i+1} = illegal(A,B,C,D,C,E) :-
not(between_file(A,B,C,D,E)).

$T_{i+1} = (T_i - C_i) \cup C_{i+1} \cup X$.

The point of this example is to sketch the treatment by the CW-GOLEM method of exceptions to the rule *illegal if the WR and the BK are on the same file*. An instance of this rule is the literal "illegal(a,2,a,6,a,4)", which is shown on the left of Figure 24. An exception to this rule, where the checking relation between the WR and the BK is blocked by the interposed WK is the literal "not(illegal(a,4,a,6,a,2))". This is shown on the right of Figure 24.

6.3.2 Results

In this experiment all positive literals in the test set were covered by the theory induced at step 1. The only exceptions were negative literals found to be covered

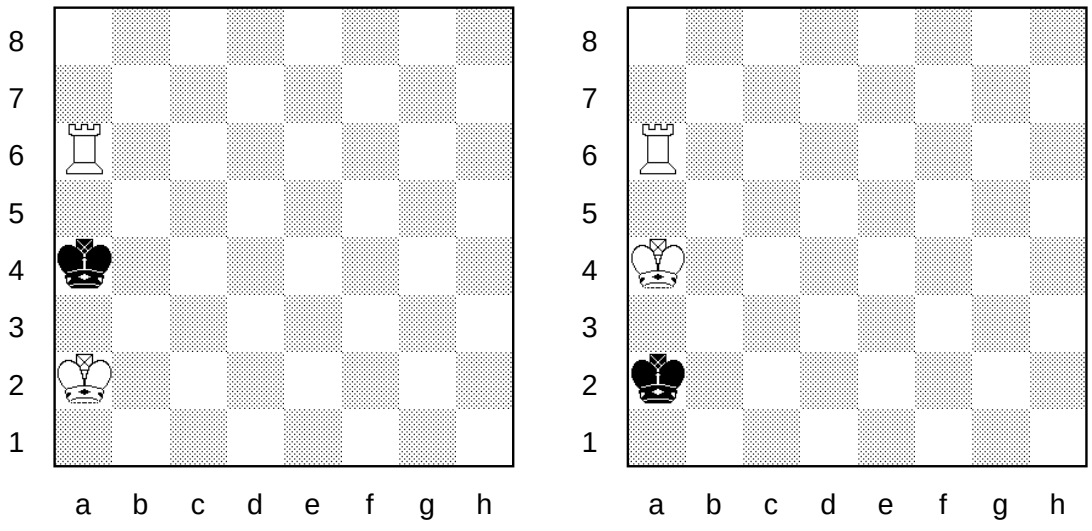


Figure 24: Two literals covered by the rule "illegal(A,B,C,D,C,E)". On the left, the position is illegal since BK:a4 is in check with WTM. On the right, the position is legal since WK:a4 blocks the attack of WR:a6 on BK:a2. The rule that positions are illegal if the WR and BK are on the same file must be specialised so as not to cover the class of positions where all pieces are on the same file with the WK between the WR and the BK.

by the theory at step 2.

Output

The theory resulting from the first generalisation step (step 1) was:

illegal(A,B,A,B,C,D).

illegal(A,B,C,D,C,E).

illegal(A,B,C,D,E,D).

illegal(A,B,C,D,E,F) :- adj(A,E), adj(B,F).

The first clause describes all cases where the White King and the White Rook are on the same square. All other possible configurations with two or more pieces on the same square are covered by the other three clauses. Clause 4 describes

all cases where the kings are adjacent. Clauses 2 and 3 cover all cases where the White Rook and the Black King are collinear, (i.e. on the same file or rank), which includes those with the Black King in check.

Clearly, this logic program is complete; all illegal positions are classified as illegal. However, it is incorrect since positions in which the three pieces are collinear but the White King is in between the White Rook and the Black King are classified as illegal by clauses 2 and 3. Following the successful specialisation of the incorrect clauses (step 3a) and generalisation of the exceptions (step 3b), the resultant theory was:

```
illegal(A,B,A,B,C,D).
illegal(A,B,C,D,C,E) :- not(between_file(A,B,C,D,E)).
illegal(A,B,C,D,E,D) :- not(between_rank(A,B,C,D,E)).
illegal(A,B,C,D,E,F) :- adj(A,E), adj(B,F).

between_file(A,B,A,C,D) :- lt(B,C), lt(D,B).
between_file(A,B,A,C,D) :- lt(B,D), lt(C,B).
between_rank(A,B,C,B,D) :- lt(A,C), lt(D,A).
between_rank(A,B,C,B,D) :- lt(A,D), lt(C,A).
```

Two new predicates have been invented and generalised to cover the exceptions to the White Rook and Black King collinearity rules. The user-named predicate "between_file" applies to file collinearity exceptions. Similarly, "between_rank" applies to rank collinearity exceptions. The definitions of both require all pieces to be on the same file or rank and impose the ordering that the White King is in between the White Rook and Black King. However, without generalising the exceptions, the parent "illegal" clauses would have remained under-specialised, (i.e., incorrect).

The predicates "adj/2" and "lt/2" were supplied as background, where the definition of the former was "adjacent-or-equal". The definition of the latter was "strictly-less-than". Both were defined only for the finite domains of the eight rows or columns of the chess-board.

Accuracy

The effect of using the more efficient induction method provided by GOLEM is that in this experiment the incremental process requires only a single cycle. Following the generalisation, testing and exception-handling steps, no exceptions were found on testing of the output theory. No exceptions were found when the theory was tested on an independent test set.

Cycle	Classification performance		Elap. (seconds)
	(a)	(b)	(c)
0	99.61	99.72	145.3
–	100.0	100.0	52.4

Key to measures :

(a) Predictive accuracy (%) on 1st test set (10000).

(b) Predictive accuracy (%) on 2nd test set (10000).

(c) Elapsed time for induction of all clauses by GOLEM, (C, Sun Sparcstation 1+).

6.3.3 Discussion

The result of 100% accuracy required the use of background predicates. In particular, "adj/2" was carefully designed for the particular problem, i.e. it is almost domain specific. It remains a goal for most inductive methods to automatically invent such utility predicates, rather than rely on the user.

Although the case does not arise in the chosen experimental domain, obviously the exceptions to any clause may themselves have exceptions, and so forth. Some of McCarthy's and Vere's examples demonstrate such levels of exceptions [59, 137]. For example, to re-express in our non-monotonic formalism some of McCarthy's axioms on the ability of things in general and birds in particular to fly:

```
thing(X) :- flightless(X).
```

```
flightless(X) :- not(non_flightless(X)).
non_flightless(X) :- not(non_non_flightless(X)).
non_non_flightless(penguin).
non_non_flightless(ostrich).
```

Although this example is rather convoluted, the phenomenon seems to occur naturally in certain rule expressions. To anticipate the next point, however, the size of any exceptions set may be very small relative to the size of the set of objects covered by the general rule. Further levels of exceptions could follow the same trend.

Clearly, incorrectness may arise in the real world not only through genuine exceptions but through noise. The number of exceptions may (as in the case of KRK illegality) be a very small proportion of the available data. The decision to treat contradictory data as noise or genuine exceptions is therefore important, but not addressed in this work.

Lastly, these results are only indicative of the potential of the techniques applied in Experiment 2. We feel that the general approach will be appropriate for many real-world problems, but empirical results from significant domains are required to back this up. Also, current implementations in Prolog of CWS are rather inefficient, although if on average we only expect small numbers of exceptions to any clause in our theory then this may not be a problem.

6.4 Summary

In this chapter we have presented first results from the application of the Closed-World Specialisation technique developed in Chapter 5. The task domain was again KRK illegality. The KRK illegality problem has been employed by previous authors to demonstrate the performance improvement available when learning in first-order formalisms compared to propositional formalisms. However,

in their solutions the improvement was purchased at the expense of incorrectness, i.e. over-generalisation. The induction of a 100% correct solution for KRK illegality presented in this chapter was dependent not solely on efficient generalisation using a first-order hypothesis language, but additionally on its combination with a method of minimal specialisation.

To our knowledge the original publication of this result [3] was the first time a complete and correct solution to this standard problem had been recorded by an incremental learning system. This advance was possible due to efficient generalisation in a first-order hypothesis language, and minimal specialisation in a non-monotonic framework. We note that both these aspects of the system were critically dependent on improvements in the knowledge representation used by the learning components. In the next chapter we report on an application of the same learning system to a more taxing problem in the same domain, namely to learn classification rules for optimal KRK strategies.

Chapter 7

Learning Optimal KRK Strategies

Move-perfect databases for chess endgames can be constructed by full-width backup from won positions. Complete tabulations are available for certain endgames, such as King and Rook against King (KRK), which contain optimal depth-to-win information. Previous work has applied decision-tree learning to construct rules for the classification of positions from such databases as won or not won, but with limited success. The current work takes an Inductive Logic Programming approach, using methods of generalisation in first-order logic and specialisation by predicate invention. Results are given on learning rules from example black-to-move KRK positions which are won-for-white in a fixed number of moves.

7.1 Motivation and First Results

Could a machine learn to play a simple chess endgame *optimally* given only example positions and some simple facts about the geometry of the board ? Below is a chess program, in Prolog, machine-learned from examples, which could legitimately form part of a feasibility demonstration for such a project.

```
krk(0,Kfile,3,WRfile,1,Kfile,1) :-
        not(wrfile_threat(Kfile,WRfile)).
krk(0,c,WKrank,a,WRrank,a,BKrank) :-
        wrrank_safe(WKrank,WRrank,BKrank).

wrfile_threat(BKfile,WRfile) :- diff(BKfile,WRfile,1).

wrrank_safe(2,WRrank,1) :- lt(2,WRrank).
wrrank_safe(Krank,WRrank,Krank) :- diff(Krank,WRrank,2).
wrrank_safe(Krank,WRrank,Krank) :- diff(Krank,WRrank,3).
wrrank_safe(Krank,WRrank,Krank) :- diff(Krank,WRrank,4).
wrrank_safe(Krank,WRrank,Krank) :- diff(Krank,WRrank,5).
wrrank_safe(Krank,WRrank,Krank) :- diff(Krank,WRrank,6).
wrrank_safe(1,8,1).
```

This program is complete and correct for the canonical set of legal Black-to-move (BTM) positions in the King and Rook against King (KRK) endgame which are won-for-White (wfw) at a depth of 0 moves (i.e. checkmate). User-supplied mnemonic names for variables and machine-invented predicates have been substituted throughout. The top-level predicate is **krk/7**, the first argument of which gives the depth of win in moves for a minimax-optimal strategy. The other six arguments specify positions by the file and rank (x and y) chessboard coordinates of, respectively, the White King, White Rook and Black King. The two clauses for the top-level predicate are complete and correct in the sense that they cover all and only those positions in an exhaustive database (described in Section 7.2.1) which are won at depth 0. This solution, which is discussed in more detail in Section 7.4.2, calls the primitive predicates **lt/2** and **diff/3** which define the relations $<$ and symmetric difference for files and ranks. Apart from these background predicates, two machine-invented predicates which are here labelled **wrfile_threat/2** and **wrrank_safe/3** complete the definition.

Previous attempts to learn classification rules in chess endgame domains have typically used a much more powerful set of domain features to describe example positions than is the case in the current study. Quinlan [106] has reported that in an application of ID3 to learning a classification for the "lost n-ply" relation in the King and Rook against King and Knight (KRKN) endgame the main obstacle to producing a complete and correct set of classification rules was the lack of suitable attributes. The approach taken in our work is to attempt to overcome this problem by relying on the learning system's ability to change its domain representation incrementally, as demanded by the learning task.

The remainder of this chapter is made up of two main parts. We consider in Section 7.3 the induction of complete BTM position classifiers for each depth of win. Then the issue of correcting these classifiers with respect to the exhaustive database is addressed in Section 7.4. These are preceded in Section 7.2 by a review of the database of positions used in this work, and followed by a discussion of some relations to other work in Section 7.5.

7.2 Materials

In previous work [123, 80, 85, 3] chess endgame databases have been used as sources of training and testing examples for machine learning experiments. The current work employs an exhaustive database which is a complete tabulation for the KRK endgame. The entries of this database contain optimal depth-to-win values for all positions. These entries constitute examples of position classes at each depth of an optimal strategy. Our training and testing examples were randomly sampled from this database

7.2.1 Retrograde analysis of endgame databases

The retrograde analysis method for generating chess endgame databases as described in Chapter 2 employs reduction of the space of positions by removing

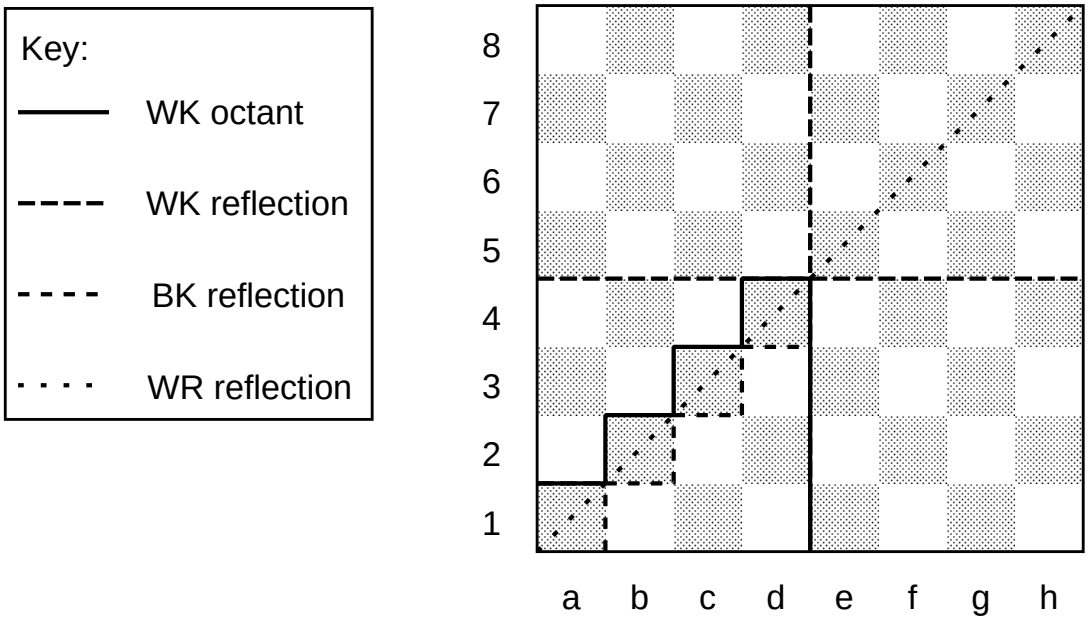


Figure 25: Canonical KRK positions.

WK octant – the ten squares {a1, b1, c1, d1, b2, c2, d2, c3, d3, d4} are the canonical locations for the White King (WK).

WK reflection – reflection about the axes indicated places the WK in a canonical location:

if the WK is above the central horizontal (rank 5 or greater), reflect the WK below (into the lower half of the board);

if the WK is right of the central vertical (file e or greater), reflect the WK to the left of (into the left half of the board);

if the WK is above the diagonal a1 to h8, reflect the WK below this diagonal (into the WK octant).

BK reflection – with the WK on squares a1, b2, c3 or d4 and the Black King (BK) above the diagonal a1 to h8, reflect the BK about this axis to place it below the diagonal.

WR reflection – with both WK and BK on the diagonal a1 to h8 and the White Rook (WR) above the diagonal, reflect the WR about this axis to place it below the diagonal.

Table 2: Depth of win, optimal play.
Depth is depth-of-win for white in moves with black-to-move; Number is number of positions. There are a total of 25260 wins. Together with 2796 draws this gives a total of 28056 positions used for positive and negative example sets.

Depth	Number	Depth	Number
0	27	9	1712
1	78	10	1985
2	246	11	2854
3	81	12	3597
4	198	13	4194
5	471	14	4553
6	592	15	2166
7	683	16	390
8	1433	Total	25260

from consideration those positions equivalent to a canonical set by symmetry. Consequently, any legal position which could be encountered for example in over-the-board play must be translated to its canonical equivalent before its database value may be retrieved. The exact symmetries which may be exploited vary according to the pieces in the endgame. In the KRK database used to provide examples for the current work three types of symmetrical translation were applied. These are diagrammed in Figure 25, which is an extension of Figure 7 in Chapter 2.

Only information on black–to–move (BTM) positions was extracted from the database. However this is sufficient to allow optimal play using only a legal move generator which is operated with 2-ply (1 move) lookahead. Since every won position is tagged in the database with its minimax-optimal depth-of-win value, and the learned definitions contain the same values, this method holds also to allow the output of the learning-from-examples method to play optimally.

7.2.2 BTM WFW positions in the KRK database

By the removal of redundancy due to symmetries the total space of legal

canonical positions in the KRK endgame is reduced from a potential 262144 to 28056. In the BTM database used in our experiments the number of legal positions won-for-white was 25260. Each of these positions is tagged with its depth-of-win value. The number of positions in each depth-of-win class is given in Table 2.

7.3 Induction of complete classifiers (not guaranteed correct)

As can be seen from Table 2, in the KRK endgame the maximum depth of win for BTM positions is 16 moves. In this chapter it will be convenient to characterise draws as infinite-depth wins. The BTM database can therefore be partitioned by depth of win into 18 disjoint subsets. Each subset can be viewed as a separate classifier, answering database queries relating to its depth of win. We begin the task of learning optimal strategies for the endgame by treating each classifier separately. For each depth of win, the relevant subset provides the positive training examples, and the remaining subsets provide the negative training examples. The first stage is to generalise these examples. We expect generalisation to compress the positive examples, in the sense that the description complexity of the generalised examples, or hypothesis, is less than that of the original examples. It is to be expected that the gain in compression will be at the expense of introducing some incorrectness into the hypothesis. Whereas the original examples were complete and correct (in the sense discussed in Section 5.1.3 of Chapter 5) for a fixed depth of win, the hypothesis is likely to cover positions at other depths of win.

The method described in Section 7.3.1 produced separate classifiers for each depth of win. These were tested in two different ways, as described in Sections 7.3.2 and 7.3.3.

7.3.1 Induction method

The learning method treated the problem of learning an optimal KRK strategy as 18 independent sub-problems. Each depth of win was a separate sub-problem. The learning method was to generalise the example positions for each depth of win using GOLEM. This guarantees completeness but not necessarily correctness. For instance, to induce a classifier for depth of win D three example sets would be constructed. All positions won at depth D formed the set of positive examples. All other positions formed the universe of negative examples. From this universe a subset of the same size as the set of positive examples was chosen randomly (sampling with replacement) to form the set of negative examples. The background examples consisted of ground instances of the predicates "symmetric difference" (defined below in Section 7.4.1) and "strictly less than" ($<$) for all unordered pairs of the file values $\{a, \ldots, h\}$ and all unordered pairs of the rank values $\{1, \ldots, 8\}$.

For large example sets or problems involving few regularities GOLEM may fail to find a generalisation. In such cases the algorithm simply outputs the set of positive training examples as its hypothesis. This occurred once in the current work, when no generalisation was found for the depth 14 sub-problem. Although draws are treated as one of the depths of win in our framework, no separate classifier was induced for this sub-problem. By default, therefore, in the testing phase the set of positive training examples consisting of all drawn positions was used as the corresponding hypothesis. Consequently, the classifiers for depth 14 and draw were complete and correct. However, a considerable penalty is paid for these properties in terms of description complexity.

7.3.2 Testing order-independent classifiers

The first method of testing the induced classifiers treated each as a separate sub-problem. To test the classifier at depth D three example sets were constructed. All positions won at depth D formed the test set of positive examples. All other

positions formed the test set of negative examples. The background examples were as for the training stage as described in Section 7.3.1.

A classifier for depth D has two outputs, yes and no. The total number of "machine predictions" made in each of these categories for the test sets of positive and negative examples were recorded. Table 3 shows the numbers of correct and incorrect predictions, together with the *a priori* frequency distribution for comparison. A complete breakdown of the classifier predictions by depth of win is given in Appendix C.

Looking through the results in Table 3 for these independently trained "intelligent guessers", an immediate thought is: how could they be used ? It seems that although these classifiers are complete, they are so error-prone as to suggest that too high a price is being paid for the simplicity of the "completeness-only" approach. In the next section, we see that these error-rates can be somewhat reduced by the use of a simple control strategy for position classification, namely by ordering the classifiers sequentially by depth from 0 upwards. Nonetheless, the inherent asymmetries of the completeness-only approach remain. It provided motivation for the closer look at complete-and-correct classifiers which ends this chapter.

7.3.3 Testing sequentially-ordered classifiers

The second method of testing the induced classifiers treated all eighteen as ordered sub-problems. To test the classifier at depth D three example sets were constructed. All positions won at depth D formed the test set of positive examples. All positions won at depth greater than D formed the test set of negative examples. The background examples were as for the training stage as described in Section 7.3.1.

A classifier for depth D has two outputs, yes and no. The total number of "machine guesses" made in each of these categories for the test sets of positive and negative examples was recorded. In Table 4 some of these results are shown.

Table 3: Numbers of correct and incorrect predictions by depth of win. Depth is the depth in moves of a win for white with black-to-move; Correctly Predicted is the number of positions correctly classified as won at Depth; Incorrectly Predicted is the number of positions incorrectly classified as won at Depth; Total Predicted (not shown in table) is Correctly Predicted plus Incorrectly Predicted; Grand Total is the number of all won and drawn positions in the database (28056); Frequency Distribution is the percentage of positions in the database which are won at Depth. Classifiers for depth 14 and draw are the respective sets of ungeneralised training examples (see text for details). Sum of percentages in column four is not equal to 100% due to rounding errors.

Depth	Correctly Predicted	Incorrectly Predicted	Frequency Distribution
0	27	99	0.10
1	78	490	0.28
2	246	238	0.88
3	81	739	0.29
4	198	1422	0.71
5	471	2961	1.68
6	592	5708	2.11
7	683	8761	2.43
8	1433	6797	5.11
9	1712	13554	6.10
10	1985	15025	7.07
11	2854	13372	10.17
12	3597	15451	12.82
13	4194	14174	14.95
14	4553	0	16.23
15	2166	11420	7.72
16	390	2593	1.39
draw	2796	0	9.97
Total	28056	112804	100.01

Table 4: Incorrectness of sequentially-ordered BTM classifiers by depth of win. Depth is the depth in moves of a win for white with black-to-move; Correctly Predicted is the number of positions correctly classified as won at Depth; Total Predicted (not shown in table) is the total number of positions classified as won at Depth; Incorrectly Predicted is Total Predicted less Correctly Predicted. Description Complexity is the number of Prolog clauses in each classifier. Classifiers for depth 14 and draw are the respective sets of ungeneralised training examples (see text for details).

Depth	Correctly Predicted	Incorrectly Predicted	Description Complexity
0	27	99	2
1	78	433	3
2	178	178	5
3	48	418	4
4	63	1098	8
5	181	2022	9
6	67	2793	9
7	90	3427	11
8	76	1118	23
9	183	4441	23
10	221	2276	25
11	143	634	53
12	359	2517	77
13	473	1696	109
14	1163	0	4553
15	1146	410	22
16	0	0	12
draw	0	0	2796
Total	4496	23560	7744

Inspection of Table 4 reveals that the classifiers under the sequential-ordering test scheme appear to be performing better (less incorrectly) than those under the order-independent method. However, some caution is required in interpreting these results, since the order-independent classifiers *each* classify all 28056 positions, whereas the sequentially-ordered classifiers *together* classify the positions. Viewed another way, the order-independent classifiers together classify 18 times as many positions as the sequentially-ordered classifiers.

An associated feature is that the sequentially-ordered set delivers a unique class value for each of the 28056 target objects. The unordered collection of independent classifiers (Table 3) assigns to each object several alternative values. In the absence of further information, a hypothetical user of the unordered set can do no better than make a random selection from the set of class values associated with each object.

Table 4 also highlights in column four the penalty paid in terms of description complexity by the depth 14 and draw classifiers. Although both complete and correct, they are more than two orders of magnitude greater in complexity (4553 and 2796, respectively) than most of the other classifiers.

Issues of completeness and correctness are manifested in greater detail in Table 5. The completeness of all classifiers is clearly illustrated, in that no classifier for depth D fails to account for all positions it sees which are in fact won in D moves. Also clearly apparent from this confusion matrix is the fact that the depth 14 classifier is complete and correct. Slightly disappointing is the result that the level of incorrectness for depth < 16 classifiers is such that all 390 positions won optimally in 16 moves are misclassified, primarily by the depth 15 and depth 13 classifiers.

A similar effect is seen for draws, but with a striking difference. Whereas for other depths most misclassifications of positions won in D occur in classifiers for $D-1$, $D-2$, in other words close to D, with draws the misclassification profile is much more evenly spread. The number of misclassified draws peaks at 455 with the depth 6 classifier. A possible reason for this is that most draws in

Table 5: Confusion matrix for sequentially-ordered BTM classifiers.

The ordering of the complete classifiers results in zeroes for cells above the leading diagonal. All positions won at depth 16 or drawn are incorrectly classified as won at depths less than 16, and hence left nothing for the depth 16 classifier to work on. Note that the pattern of misclassification of draws differs from other depths of win. See text for further details.

Actual depth of win from BTM database against depth of win predicted by BTM classifiers

Actual \ Predicted	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	draw	Total
0	27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27
1	0	78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	78
2	0	68	178	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	246
3	0	1	32	48	0	0	0	0	0	0	0	0	0	0	0	0	0	0	81
4	0	3	26	106	63	0	0	0	0	0	0	0	0	0	0	0	0	0	198
5	0	6	26	178	80	181	0	0	0	0	0	0	0	0	0	0	0	0	471
6	0	16	13	27	160	309	67	0	0	0	0	0	0	0	0	0	0	0	592
7	0	33	16	24	69	245	206	90	0	0	0	0	0	0	0	0	0	0	683
8	0	23	6	41	147	432	451	257	76	0	0	0	0	0	0	0	0	0	1433
9	4	59	1	11	83	373	330	569	99	183	0	0	0	0	0	0	0	0	1712
10	11	68	6	6	75	164	379	623	122	310	221	0	0	0	0	0	0	0	1985
11	16	63	10	0	118	198	344	678	174	805	305	143	0	0	0	0	0	0	2854
12	20	60	17	0	92	112	265	571	209	943	761	188	359	0	0	0	0	0	3597
13	10	9	9	0	71	35	212	289	120	1057	432	229	1248	473	0	0	0	0	4194
14	6	0	0	0	58	5	147	218	78	874	332	57	846	769	1163	0	0	0	4553
15	0	0	0	0	14	0	4	27	30	148	175	17	128	477	0	1146	0	0	2166
16	0	0	0	0	0	0	0	0	1	3	37	0	1	121	0	227	0	0	390
draw	32	24	16	25	131	149	455	195	285	301	234	143	294	329	0	183	0	0	2796
Total	126	511	356	466	1161	2203	2860	3517	1194	4624	2497	777	2876	2169	1163	1556	0	0	28056

the domain are due to the white rook being *en prise*. These are likely to contain positional patterns quite similar to those in won positions, where the rook is safe.

Figure 26 is inconclusive regarding the relationship of predictive accuracy to description complexity if the outliers (those with 100% accuracy but high complexity) are not considered. However, the measure of complexity being used here is probably too simplistic to judge the relationship satisfactorily. There is nonetheless an interesting pattern in the main cluster, where predictive accuracy steadily falls and complexity increases from depth 0 to depth 9. Thereafter the pattern reverses, with accuracy increasing and complexity decreasing from depth 10 to depth 16. This profile over the (approximate) ordering by depth of the classifiers reflects the trend descending columns one, three and four in Table 4 (disregarding depth 14 and draw classifiers).

Figure 27 is intended to present information which is already implicit in Figure 26 in a slightly different way. For example, it shows, promisingly (from the point of view of automatic concept formation), that 13 out 18 classifiers are constructed from 32 or fewer Prolog clauses.

7.3.4 Summary

From Table 5 we can see that since all classifiers are complete, there are no positions predicted as won in D moves which are actually won in $< D$ moves. This is because any position which is in fact won at depth D will be correctly identified as such by the classifier for depth D. For each depth D classifier (except for depths 14, 16 and draw) there are many positions actually won at depth $> D$ which are predicted as won optimally in D moves. This illustrates the incorrectness of these classifiers.

Recall that any position to be classified under this testing regime is tested against the classifier for each depth, in ascending order from depth 0. Comparing the mean number of positions misclassified under the order-independent method

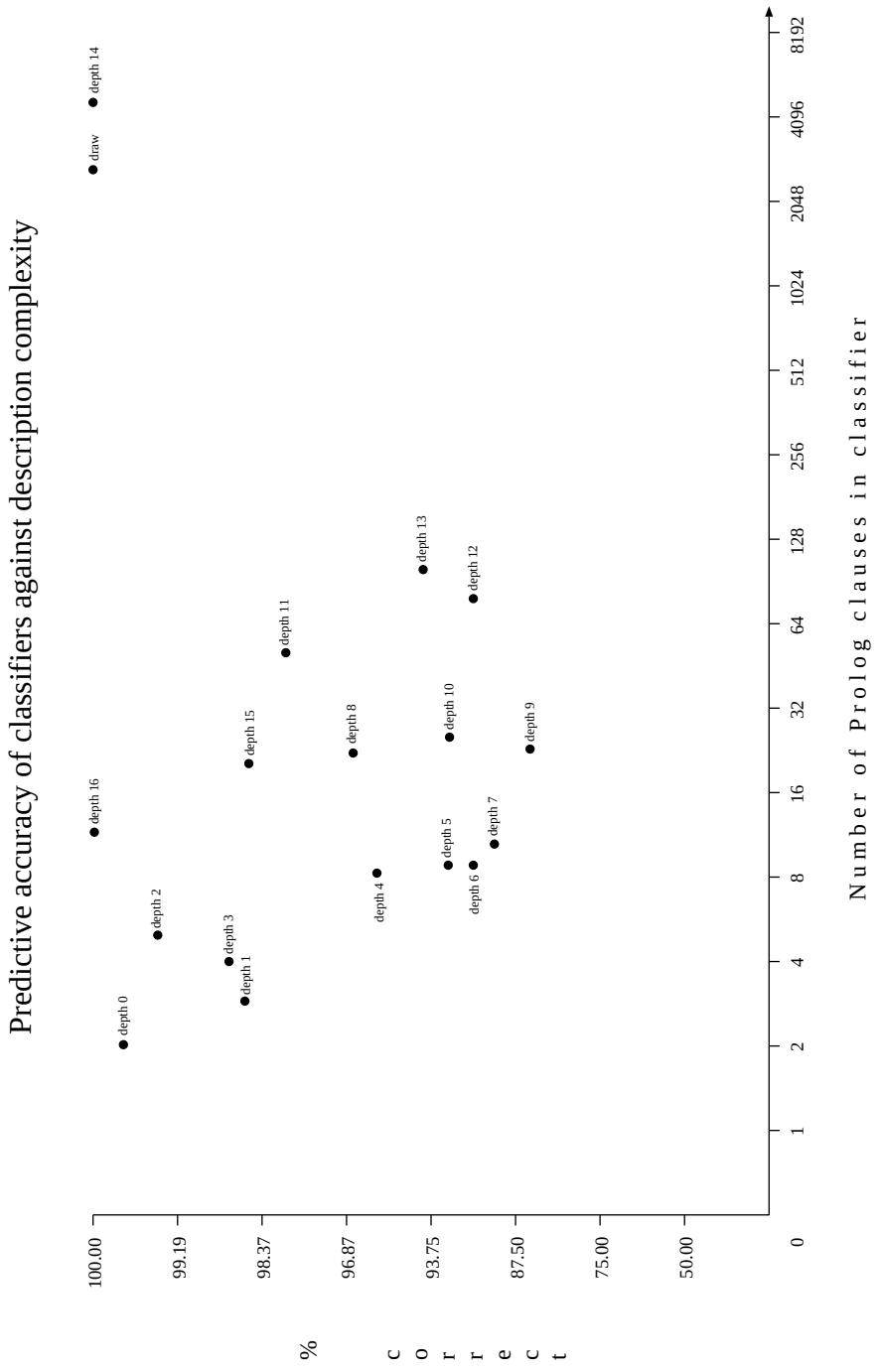


Figure 26: Predictive accuracy against description complexity for sequentially-ordered BTM classifiers.

Note: log scale used on both axes. For each classifier, accuracy is the percentage of all 28056 positions which are not incorrectly predicted. Description complexity is the number of Prolog clauses in a classifier. Classifiers for depth 14 and draw contain only ground examples. See text for further details.

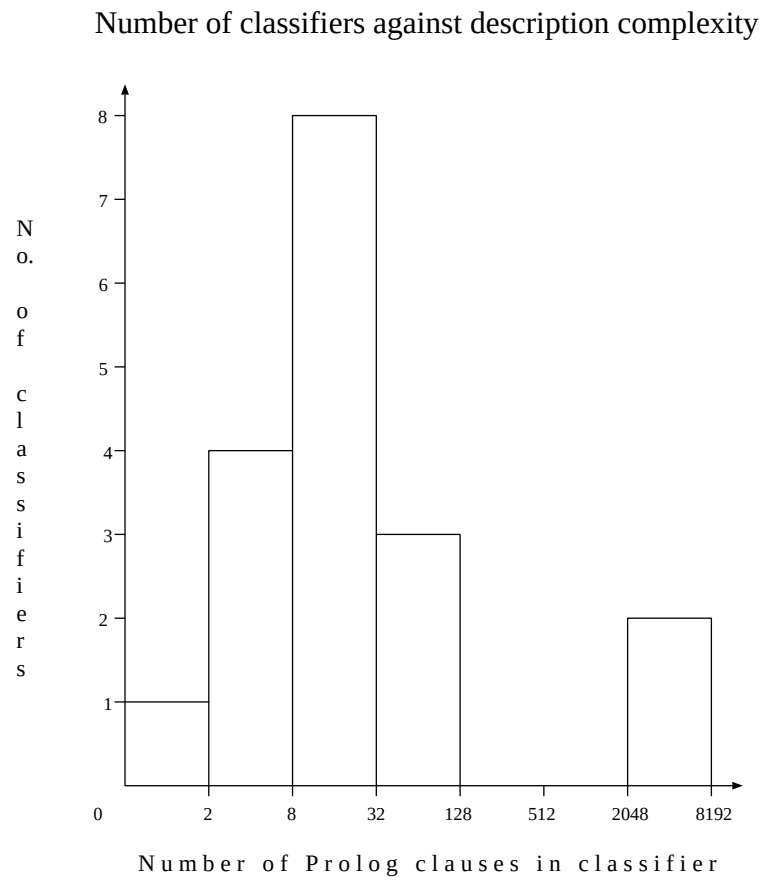


Figure 27: Histogram showing number of classifiers in description complexity bands.

Note: log scale used on horizontal axis for description complexity bands of classifiers, measured by number of Prolog clauses in classifiers.

with the total number of positions misclassified under the Prolog-style control of the sequentially-ordered classifiers, the classification performance is improved in the latter approach. The more sophisticated control of the sequentially-ordered classifiers improves the classification performance in comparison with the order-independent method. The former records the performance of each classifier as an independent agent. The latter records the performance of every classifier acting cooperatively, as ordered by an external controller. Declaratively, however, the respective classifiers in both cases are, viewed as concept descriptions, identical.

In view of the obvious incorrectness of the induced theory in this work, it is worth recalling that the problem we are attacking is a difficult one. That this is so is apparent from a past failed attempt made by unaided human experts to construct by analysis an optimal classifier theory. Clarke [20] established (by constructing a database) that, contrary to previously published statements in standard texts, KRK was won for white in a maximum of sixteen moves.

7.4 Induction of complete and correct classifiers

From the results of the previous section it is notable that although complete and in most cases compressed (more concise than the training examples) the classifiers at all depths of win where generalisation took place are incorrect. This seems a suitable test, therefore, of the specialisation methods developed in this thesis. In this section we present results for complete and correct Prolog programs for BTM positions won optimally in 0 and 1 moves, respectively.

7.4.1 Method

In this work the goal is the induction of complete, correct and concise theories in the KRK domain. Example positions for Black-to-move (BTM) and win at depth D, where D is a number of moves, were extracted from an exhaustive

database computed by the standard retrograde analysis method as discussed above. In this method, each entry in the database contains a position labelled by its minimax-optimal game-theoretic value, which in this case is its depth D. In logical terminology such a database may be thought of as an extensional definition of the optimal play of the endgame. Therefore the database is a complete and correct theory of KRK. Alternatively, the database is a many-one relation $< P, D >$ where P is a canonical position and D is depth-of-win. However, despite reductions due to the removal of positions equivalent by symmetry, the database is too large to be called a concise theory. The goal was therefore the induction of a program which:

1. on input of a legal BTM position in KRK outputs the minimal depth-of-win for white;

2. on some measure was more concise than the database representation.

In this section the experimental tasks were restricted to positions with depth 0 or 1. Also, we did not apply any measure of relative conciseness [1]. A suitable candidate measure could be HP-compression as described in [79].

We adopted an Inductive Logic Programming approach, using a new algorithm called GCWS ("Generalising Closed-World Specialisation"). GCWS is a combined generalisation and specialisation system. It carries out incremental learning, and incorporates theory-guided sampling. An algorithm schema for GCWS is given in Figure 28. The generalisation phase of GCWS appears in Figure 28 as a function call "Gen(training, background)". Generalisation is described in Figure 29. The specialisation phase of GCWS appears in Figure 28 as a function call "Spec(P', test, background)". Testing and specialisation are described in Figure 30. In Figures 28, 29 and 30 a procedural pseudo-code with a C-like syntax is used to describe GCWS.

Generalisation steps are carried out in GCWS using a version of Relative Least General Generalisation (RLGG) as implemented in Muggleton and Feng's

[1] See discussion in Section 7.5.

Algorithm GCWS

Input : P_i, training, test, background

  if training $= \emptyset$ then

    $P_f = P_i$

  else

    $P = \text{Gen}(\text{training}, \text{background})$

    $P' = P_i \cup P$

    $\langle P'', \text{exceptions} \rangle = \text{Spec}(P', \text{test}, \text{background})$

    $P_f = \text{GCWS}(P'', \text{exceptions}, \text{test}, \text{background})$

Output : P_f

Figure 28: GCWS algorithm schema.

GOLEM system [87]. GOLEM appears in the generalisation procedure schemas of Figure 29 as a function called with four arguments. These are, respectively, the number of negative examples (i.e. exceptions) allowed to be covered by any hypothesis, a set of positive examples, a set of negative examples and a set of background examples.

Correction of over-general clauses with respect to a target model was carried out using the Closed-World Specialisation (CWS) technique developed in Chapter 5 in a new Prolog implementation by the author. CWS appears in the specialisation procedure schema of Figure 30 as a function called with two arguments. These are, respectively, the program to be specialised, and a set of negative examples selected from the test set by theory-guided sampling.

GCWS is based on CW-GOLEM, described in Chapter 6. It extends CW-GOLEM by allowing the automatic invention and introduction of non-negated exception predicates during the specialisation process. The GCWS method can

Procedure Gen

Input : $\langle E^+, E^- \rangle$, background

$x = 0$

P = GOLEM(x, E^+, E^-, background)

if $|P| \geq 10$ then

P = OverGen(1, $\langle E^+, E^- \rangle$, background)

Output : P

Procedure OverGen

Input : x, $\langle E^+, E^- \rangle$, background

P = GOLEM(x, E^+, E^-, background)

if $|P| \geq 10$

and $x < |E^-|$ then

$x *= 2$;

P = OverGen($x, \langle E^+, E^- \rangle$, background)

Output : P

Figure 29: Generalisation procedure schemas.

Procedure Spec

Input : P, test, background

/* Theory-guided sampling */

$E^+ = \{e | e \in \text{test}, P \vdash e\}$

$E^- = \{\bar{e} | \bar{e} \in \text{test}, P \vdash e\}$

/* Batch specialisation */

$\langle P', \text{pos\_exc} \rangle = \text{CWS}(P, E^-)$

/* Negative exceptions */

$\text{neg\_exc} = \text{pos2neg}(P', E^+)$

/* Exception folding */

if $|\text{pos\_exc}| \leq |\text{neg\_exc}|$ then

  exceptions = $\langle \text{pos\_exc}, \text{neg\_exc} \rangle$

else

  exceptions = $\langle \overline{\text{neg\_exc}}, \overline{\text{pos\_exc}} \rangle$

Output : $\langle P', \text{exceptions} \rangle$

Figure 30: Specialisation procedure schema.

be viewed as implementing a form of automatic hierarchical problem decomposition.

An informal complexity constraint on the number of clauses used in any predicate definition was applied for learning the depth 0 and 1 definitions, based on the hypothesised limit on human short term memory capacity of 7 ± 2 chunks. (A Prolog predicate is defined using one or more Horn clauses, where each clause contains a single positive literal with the same predicate symbol and arity).

To avoid bias of the method by supplying the learning algorithm with a large number of predefined domain-specific background predicates, the background knowledge was restricted to contain only one specifically chess-oriented geometrical predicate, namely the symmetric difference between files and between ranks. This is defined as follows: for two file (resp. rank) values V_1 and V_2 the symmetric difference is $\mathrm{abs}(V_1 - V_2)$, where $V_1, V_2 \in \{1, .., 8\}$ and $\mathrm{abs}(X)$ is X if $X \geq 0$ or $0 - X$ otherwise. The other background predicate available was "strictly-less-than" ($<$) over files and over ranks.

These background predicates were selected as basic building blocks for the expression of piece relations in terms of the geometry of the chess board. In particular, they facilitate expression of the types of attack and counter-attack relationships which would seem to be essential for the expression of higher-level chess concepts such as capture, safety, check, etc. Thus they supply some of the raw materials for relevant predicate invention in chess domains.

Details of the experimental method are given in Figure 28 in the form of an algorithm schema. At each depth of win, the positive examples are all positions in the database which are won for white at that depth, and the negative examples are selected randomly from the remaining positions in the database.

Each depth of win was treated as a separate learning problem. The results for depth 0 are in Section 7.4.2 and for depth 1 in Section 7.4.2.

7.4.2 Results

To recap on the Prolog representation used, the target predicate for the concept was **krk/7**. The first argument of this predicate indicates depth of win, in the range 0 to 16 (although only 0 and 1 were used in the present study). The remaining six arguments give the file and rank coordinates for, respectively, the White King, the White Rook and the Black King. File arguments are in the range a to h while rank arguments are in the range 1 to 8. Note that these values are those used in the standard algebraic chess notation. For instance, the unit clause "krk(0, c, 1, a, 3, a, 1)" is read as "the black-to-move position WK:c1 WR:a3 BK:a1 is won-for-white at depth 0 (i.e. in 0 moves)". Recall that only legal positions are considered.

Background knowledge was restricted to the predicates **diff/3**, symmetric difference, and **lt/2**, strictly less than. Two argument types were used for file and rank arguments, as for the target predicate. A third type was used for symmetric difference arguments.

Prolog programs induced for depths 0 and 1 are presented in Figures 31 and 34. In these figures a key to machine-invented predicates is provided to improve readability. Throughout this section we illustrate each top-level clause in the induced programs with a figure containing diagrams of partial chessboards. The diagrams indicate position classes covered by the clause indicated. Dotted lines and arrows are used to show variations in placing the attacking White Rook.

BTM WFW depth 0

The induced Prolog definition for BTM WFW depth 0 is shown in Figure 31. Clause 1 of this definition is illustrated in Figure 32. As in clause 1, when the new predicate has only a single clause in its definition, it may be replaced with the clause body. In incremental learning this could however be undesirable – the invented predicate if retained might have further clauses added to its definition. For the time being we can interpret clause 1 as follows : "any BTM

```
krk(0,A,3,B,1,A,1) :- not(nonkrk1(A,B)).      % Clause 1
nonkrk1(A,B) :- diff(A,B,d1).

krk(0,c,A,a,B,a,C) :- krk2(A,B,C).            % Clause 2
krk2(2,A,1) :- lt(2,A).
krk2(A,B,A) :- diff(A,B,d2).
krk2(A,B,A) :- diff(A,B,d3).
krk2(A,B,A) :- diff(A,B,d4).
krk2(A,B,A) :- diff(A,B,d5).
krk2(A,B,A) :- diff(A,B,d6).
krk2(1,8,1).

%  Key (labels for machine-invented predicates; background predicates) :
%    nonkrk1(A,B) = "wrfile_threat(WKf,WRf)"
%    krk2(A,B,C) = "wrrank_safe(WKR,WRr,BKr)"
%    diff(A,B,d1) = "symmetric difference between A and B is 1"
%    lt(2,A) = "2 < A"
```

Figure 31: BTM WFW depth 0

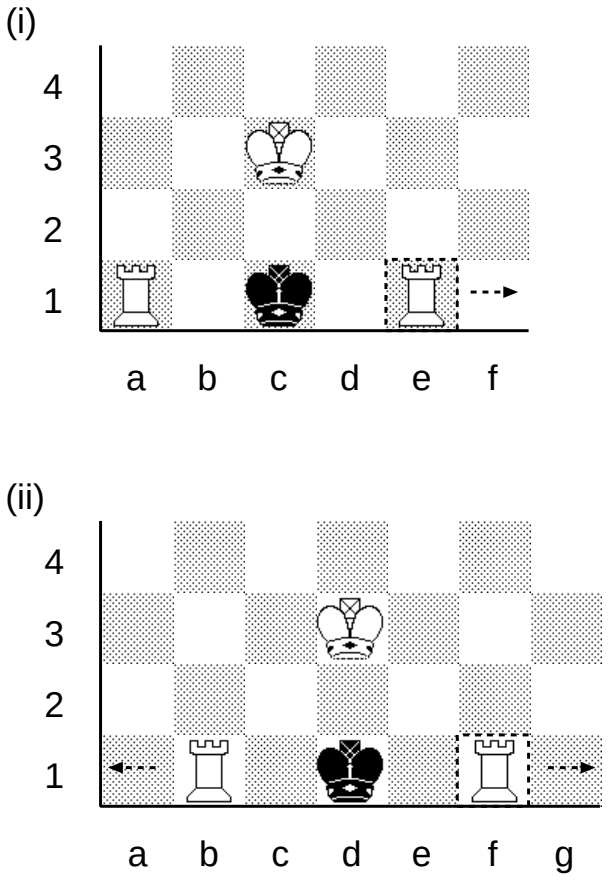


Figure 32: BTM WFW depth 0, clause 1

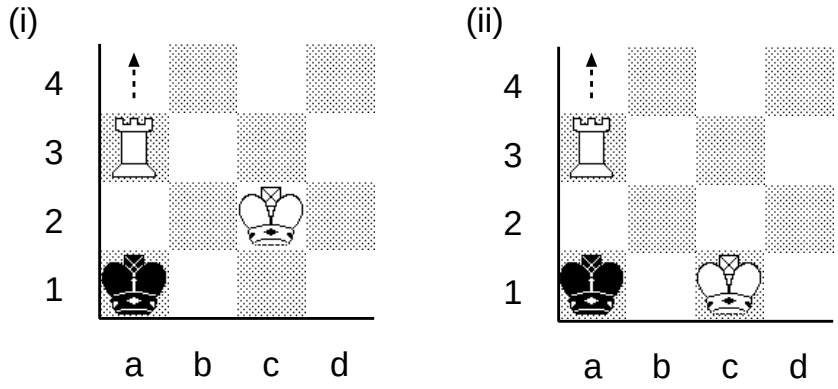


Figure 33: BTM WFW depth 0, clause 2

KRK position with the kings on the same file, the white king on rank 3 and both the rook and black king on rank 1 is WFW depth 0 when the symmetric difference between king and rook files is not 1". The machine-invented predicate **nonkrk1/2** clearly relates to the idea of rook safety when the black king is in check. It might be labelled **wrfile_threat/2**. This concept is apparent from the diagrams in Figure 32. Diagram (i) shows the placings of the White Rook on rank 1, essentially on file a and files e to h. Diagram (ii) illustrates the same pattern shifted one file to the right, with the White Rook on rank 1, files a, b and f to h.

The second clause is in a sense the dual of the first, this time for rank values, with the machine-invented predicate **krk2/3** being labelled **wrrank_safe/3**. This is shown in Figure 33. Diagram (i) illustrates the first clause of **krk2/3**, with the Kings not in opposition. Diagram (ii) covers the remaining clauses of **krk2/3** where the Kings are in opposition. In all cases the White Rook attacks the Black King down file a from the safety of rank 3 or above.

These two clauses cover all 27 positions won at depth 0. Although this is a small number of examples, the complexity of describing the concepts involved is clear from the diagrams of Figures 32 and 33.

BTM WFW depth 1

The induced Prolog definition for BTM WFW depth 1 is shown in Figure 34. The top-level predicate **krk/7** has three clauses in this definition. Clause 1 of this definition is illustrated in Figure 35. There are four diagrams in this figure, each of which show essentially the same pattern of attacking relationships between the pieces. Diagrams (i) to (iv) depict the patterns for the four clauses of the definition of the invented predicate **krk1/4**. This predicate might be labelled **wrrank_safe**, since it is principally a condition on the White Rook's rank safety. This is because the key attack relationships are defined by the top-level of clause 1. To see that the patterns do indeed cover checkmate at depth 1, refer to diagram (i) in Figure 35. The Black King is forced to move to c1,

```
krk(1,A,3,B,C,D,1) :-
      krk1(A,B,C,D), diff(A,B,d2), diff(A,D,d1).      % Clause 1

krk1(c,a,A,b) :- not(nonkrk12(A)).
krk1(c,e,A,d) :- not(nonkrk12(A)).
krk1(d,b,A,c) :- not(nonkrk12(A)).
krk1(d,f,A,e) :- not(nonkrk12(A)).

nonkrk12(1).
nonkrk12(2).

krk(1,c,2,A,B,a,C) :- krk2(A,B,C).      % Clause 2

krk2(A,4,3) :- not(nonkrk21(A)).
krk2(A,3,2) :- not(nonkrk22(A)).
krk2(A,4,1) :- not(nonkrk22(A)).
krk2(A,5,1) :- not(nonkrk22(A)).
krk2(A,6,1) :- not(nonkrk22(A)).
krk2(A,7,1) :- not(nonkrk22(A)).
krk2(A,8,1) :- not(nonkrk22(A)).

nonkrk21(a).
nonkrk21(b).

nonkrk22(a).

krk(1,c,1,A,3,a,2) :- not(nonkrk3(A)).      % Clause 3

nonkrk3(a).
nonkrk3(b).

%   Key (labels for machine-invented predicates) :
%     krk1(A,B,C,D) =“wrrank_safe(WKf,WRf,WRr,BKf)”
%     krk2(A,B,C) =“wrfile_safe(WRf,WRr,BKr)”
%     nonkrk12(A) =“rank_lessthan_or_equal_2(WRr)”
%     nonkrk21(A) =“file_lessthan_or_equal_b(WRf)”
%     nonkrk22(A)=“cornerfile(WRf)”
%     nonkrk3(A) =“file_lessthan_or_equal_b(WRf)”
```

Figure 34: BTM WFW depth 1

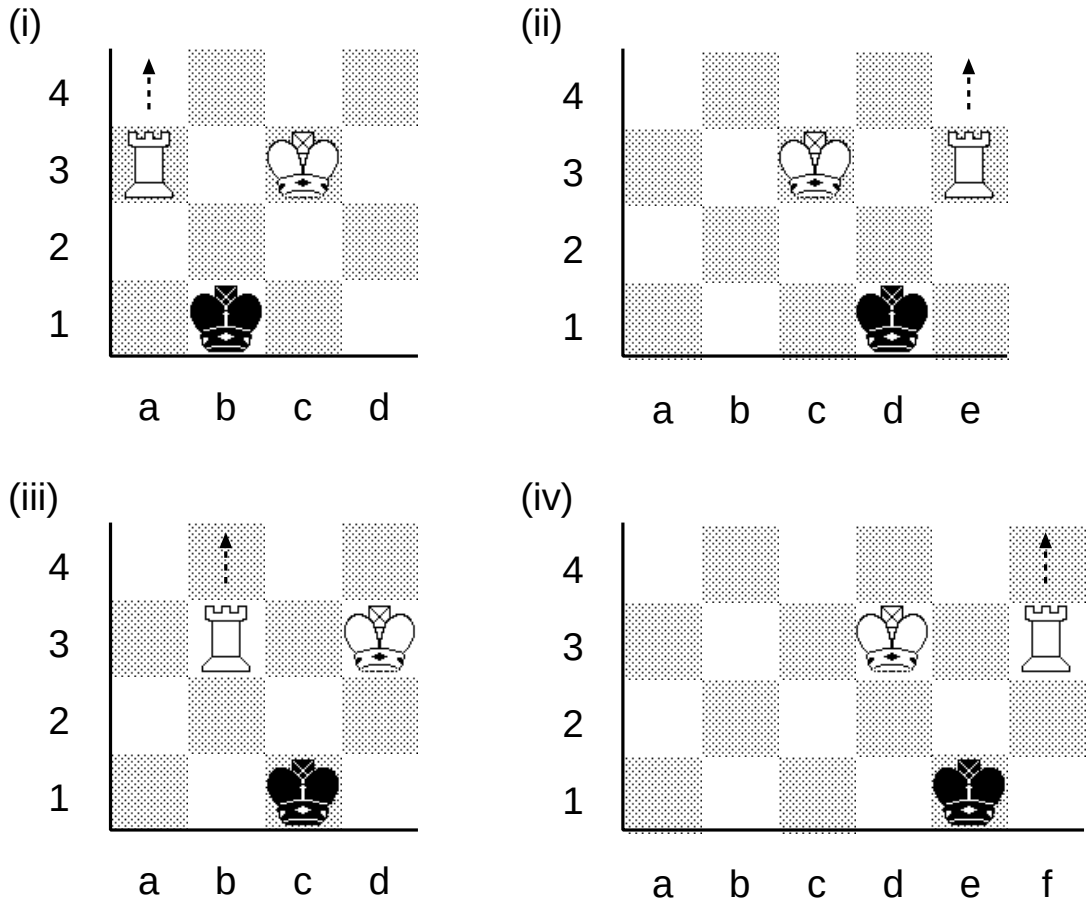


Figure 35: BTM WFW depth 1, clause 1

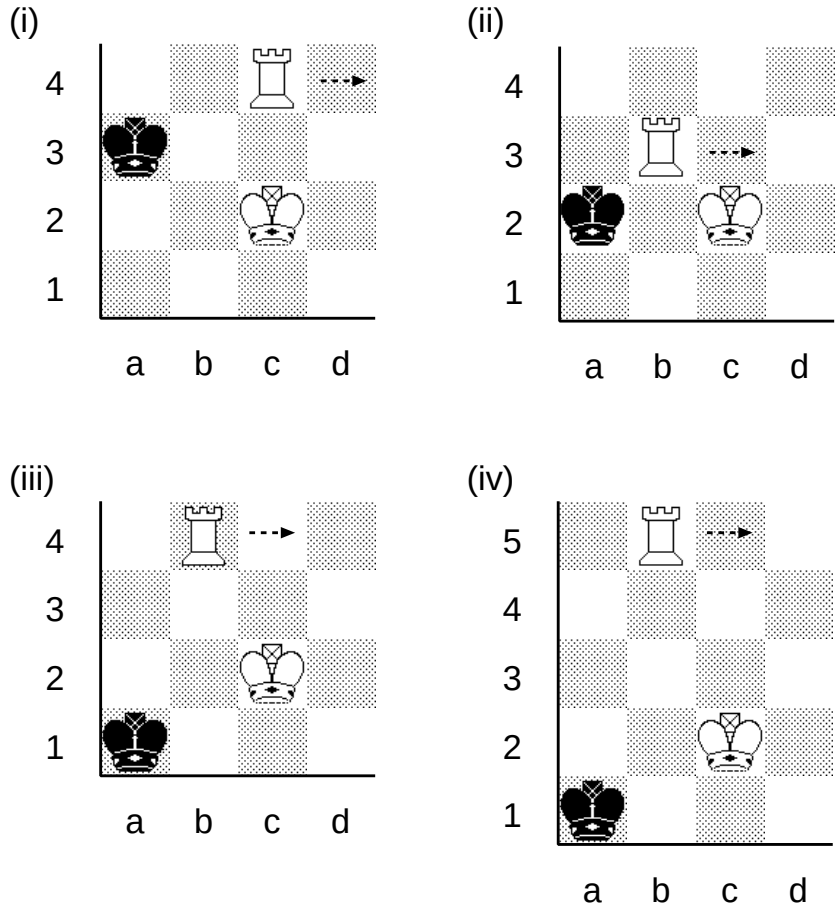


Figure 36: BTM WFW depth 1, clause 2

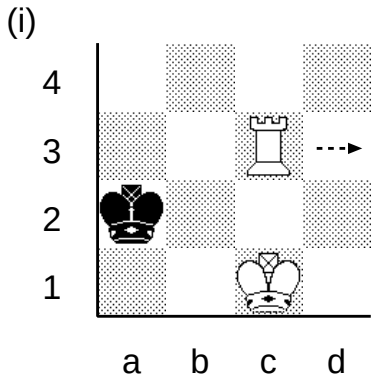


Figure 37: BTM WFW depth 1, clause 3

which is followed by the White Rook moving to a1. Black is in check with no further moves. Recall that this checkmate position is covered by the pattern of diagram (i) in Figure 32.

The second clause of the BTM WFW depth 1 definition has only the machine-invented predicate **krk2/3** in the clause body. The clause has the White King fixed on square c2, and a condition is required on the relation of the White Rook to the Black King. We might label this condition **wrfile_safe**, since the White Rook is restricting the movement of the Black King without directly attacking it or being attacked by it. The key patterns are shown in Figure 36. In diagram (i) the locations of the Kings are fixed with the White Rook free to occupy rank 4 on any file apart from a or b. This pattern corresponds to the first clause in the definition of **krk2/3**. Diagram (ii) presents an interesting configuration with the Kings in opposition. This pattern corresponds to the second clause of **krk2/3**, which covers one position where the White Rook is attacked by the Black King but defended by the White King. The remaining clauses in the definition cover positions falling into pattern classes closely related to that of diagram (i). For instance diagrams (iii) and (iv) correspond to clauses 3 and 4, with the Black King forced to move into opposition on file a followed by the the White Rook moving to give checkmate on this file.

The third clause in the definition of the predicate **krk/7** covers positions fitting the pattern of Figure 37. Here the positions of both Kings are fixed, and the Black King is forced to move into square a1. From its safe position on rank 3 at file c or greater, the White Rook moves to a3 giving checkmate. The reader may care to verify that the patterns shown account for all 78 of the canonical BTM positions which are WFW at depth 1.

7.5 Discussion

In Section 7.4.1 we stated that a long-term goal of this work was to induce programs from database examples such that on some measure the programs

were more concise than the examples. Although some work has been done on this front [79] we do not present any results in this thesis. However, the Prolog programs [2] from Section 7.4.2 are more compact on an informal "lines of code" criterion (10 lines and 21 respectively) than their representation as ground examples (28056). More importantly, these rules together with their accompanying chessboard diagrams have been verified as meaningful by chess expert and Prolog specialist Ivan Bratko.

Work on the inductive synthesis of knowledge employing the "easy inverse trick" [68] pushed the nascent technology of decision-tree induction to its limits in the late 70s and early 80s [106]. These initial results in chess endgame domains led to a variety of successful applications. The landmark KARDIO system used a deep model of the heart to synthesise shallow rules for ECG interpretation [10]. Among the rules which came out of this study were some previously undiscovered in over 200 years of cardiology. This route was also taken in an application to satellite fault diagnosis [98]. Most recently, an ILP approach in the same domain allowed the learning of significant temporal relations not expressed in the earlier solution [25].

The rôle of Machine Learning in previous work has focused on database compression. Even with decision tree induction this compression can be significant, as in the KARDIO work, which compressed 5 MBytes of ground descriptions to 30 KBytes of rules. Typically, however, in the chess endgame applications to date the bottleneck for decision-tree induction has been selection of an adequate set of attributes. For example, in experiments on the KPa7KR domain [124] most of the effort was expended on hand-crafting the attributes which capture the necessary relational features of the won/not won predicate. The novelty of the present approach lies in the application of relational learning using RLGG as implemented in GOLEM coupled with specialisation techniques based on predicate invention. This follows from earlier results with a similar approach in the simpler KRK illegality domain [3].

[2] Not including background predicates.

The predicate invention methods by which induced clauses are specialised by introducing literals into the clause body are a special case of predicate invention within Muggleton's [78] "refinement lattice" framework.

7.6 Summary

We have presented results from learning optimal strategies in the KRK endgame. Optimality is achieved through the use of examples extracted from an exhaustive database for the endgame. A complete theory (correctness not guaranteed) for black-to-move, won-for-white positions at all depths but one (0 to 16, except 14) has been learned automatically from ground instances of positions and only low-level background knowledge. The definitions for win at depths 0 and 1 have been fully specialised and were presented as a complete and correct definition of the target concept at these levels. They have been tested fully on the exhaustive example set and validated by a human expert.

Chapter 8

Conclusion

In this chapter we review the main results presented in the thesis, list some related items from the literature on knowledge representation and discuss some possible extensions to our work. A final section contains a brief summary.

8.1 Review

We began by introducing Machine Learning as a scientific discipline which can supply significant cost benefits through its commercial application. Although such benefits had been demonstrated by others in the past the approaches used seemed to depend for their applicability on certain independently necessary capabilities. This thesis has characterised these capabilities and through experimentation identified defects in some of these approaches. Based on our results we thereby devised new methods to overcome some of these limitations.

Following from this choice of study it was decided that to facilitate comparisons a number of state-of-the-art techniques ought to be compared in the same benchmark domain. Any new techniques developed following this initial phase could then be tested on the same bench-mark. We elected to work on a chess endgame since the results of previous Machine Learning studies suggested that this type of domain had provided the most demanding tasks for the algorithms tested. It is possible to concisely and exactly specify a range of learning tasks of

varying complexity in chess endgame domains. Also, the results of experiments on such tasks can be precisely measured. Specifically, the chess endgame "King and Rook against King" (KRK) was chosen and used throughout the thesis work. As it turned out this endgame proved to be sufficiently complex to distinguish critical differences between different methods. Initially we chose the task of learning some of the rules of the game, namely the classification of positions as illegal or not. Since the publication of some of our results on the illegality classification task a number of workers have adopted it as a standard problem. Subsequently, having devised a method able to complete this task successfully we turned our attention to the much harder task of learning to play optimally.

In the first stage of experimentation we focused on the Knowledge Representation issues involved in certain general-purpose inductive learning algorithms. These could be summarised as aspects of the "inductive bias" of the algorithms. For example, a decision tree induction algorithm such as ID3 requires the user to pre-define a set of attributes which are adequate for a description of the concept to be learned. This bottleneck had been avoided in earlier studies with an algorithm, known as DUCE, capable of inventing new attributes by the process of inverse resolution. However, we found that the use of this algorithm did not avoid the representation problem in our KRK illegality task. In a further experiment which involved a wider range of learning algorithms and included human subjects, we concluded on the basis of our results that the key requirement for a learning algorithm on this task was the ability to compactly represent relational sub-concepts. This required a representation language of the expressive power of first-order logic. It seemed inevitable that a similar requirement would hold for the efficient learning of more interesting concepts in chess.

The results of the experiment were also interpreted as showing the need for generalisation and specialisation in incremental learning, in the following way. CIGOL, one of the algorithms tested, was restricted to relatively small sets of training examples. Owing to its ability to form hypotheses in a first-order

logic CIGOL achieved an average performance of around 85% with training sets of between 50 to 80 examples. The other algorithms in the study required many more training examples to reach equivalent performance levels. However, given the distribution of examples in the experimental domain under random sampling, the induction of the complete target concept by CIGOL would have required much larger training sets, in fact exceeding the maximum for available implementations of this algorithm. This problem was later addressed by using another algorithm, discussed below. Nonetheless in this experiment CIGOL could never have converged to the target concept even given the capacity to handle larger training sets. This was due to the fact that it first over-generalised certain sub-concepts, and having done so was then unable to specialise when given contradictory examples.

The main theoretical contribution of this thesis is concerned with precisely this ability of specialising a first-order logic theory by incorporating exceptions. The major result is that a minimal specialisation, i.e. one avoiding over-correction, of such a theory in classical logic is in general infeasible. Instead, by moving to a non-monotonic logic, in our case by applying the Closed World Assumption, we derived an algorithm which carries out minimal specialisations. It is interesting to note that again this is ultimately an issue of Knowledge Representation.

This algorithm, called CWS (for Closed World Specialisation), is independent of any generalisation method, and in the remainder of the thesis was experimentally tested on two separate tasks within the KRK domain. It was linked with appropriate generalisers to form NM-CIGOL and CW-GOLEM respectively. The basic structure of both comprised a method of generalisation in Horn clause logic coupled with the CWS method. Both systems were experimentally tested on the KRK illegality problem.

The experimental procedure involved was stepwise, in the following sense. Firstly, the incremental algorithm combined CIGOL for generalisation and CWS for specialisation. This algorithm, termed NM-CIGOL, when tested on the KRK

illegality domain was given the best-performing theory learned by CIGOL in the earlier experiments as its initial concept of illegality, together with a large set of examples. NM-CIGOL generated its own training sets by partitioning the large set of examples with respect to its current concept, which was accordingly to be generalised or specialised. The working hypothesis was that NM-CIGOL would improve on the initial theory, although this was not the case. In fact, overall performance was reduced by a tendency to over-generalise due to the restriction on training set size.

The GOLEM algorithm was the second generalisation method used experimentally in conjunction with CWS. Like CIGOL, GOLEM's hypothesis language is first-order Horn clause logic. Unlike CIGOL, GOLEM is a batch algorithm designed to handle large sets of training examples. Combined with CWS this gave an incremental algorithm known as CW-GOLEM. When tested on KRK illegality, using much larger training sets, CW-GOLEM rapidly converged to a complete and correct theory. NM-CIGOL was eliminated from the subsequent experiments.

CW-GOLEM was renamed GCWS (for Generalising Closed World Specialisation) following minor extensions to the specialisation method, and applied to the KRK "won-in-N moves" problem. This involves learning classification rules to form part of an optimal strategy to win the game.

This work takes the following approach. An exhaustive database contains values of the depth of win for white (measured by number of moves) for all canonical black-to-move positions in the endgame. The problem is to learn a complete and correct classification theory for each of the 17 possible values of N in the "won-in-N moves" domain. To date, GCWS has been successfully tested on depths 0 and 1, as described in the foregoing.

Note that the GOLEM algorithm, which formed a component of GCWS, was also tested in stand-alone mode on the task of finding complete (although not necessarily correct) solutions to "won-in-N moves" for N ranging over $0, 1, 2, \ldots, 16$.

8.2 Knowledge Representation

The key technique developed in this thesis is Closed-World Specialisation. This method relies on non-monotonic logic. Non-monotonic logic is a representation, not a learning methodology, although the problems of reasoning in a changing world for which non-monotonic inference methods have been developed are nowhere more apparent than within the paradigm of incremental learning. A great deal of work on non-monotonic logic has been published within the area of Knowledge Representation. We cannot provide an authoritative survey of this literature. Instead we give in this section a short annotated list of references which have certain links to the issues addressed by our method of learning logical exceptions.

Classical logic representations. Before referring to non-monotonic logic representations, some notable work on specialisation in Classical Logic representations should be mentioned. Vere [137] developed methods to deal with counterexamples during induction. These are termed "counterfactuals" and they define a set of conditions which must be false if a generalisation is to be satisfied. Lassez and Marriott [49] have dealt with the problem of generalisation given a set of counter-examples using a technique of representing explicit exceptions. As discussed previously, work in Machine Learning such as that by Michalski [63] covers handling of counter-examples in a representation based on classical logic.

Logic Programming. Apart from Shapiro's seminal work (discussed in Chapter 5) on the incremental synthesis by debugging of Prolog programs, some work has been done with similar motivations in Logic Programming. The problem is described from the viewpoint of Knowledge Assimilation for logic databases, and covers extensions to Logic Programming to allow belief revision. The key to these techniques is usually reliant on the

knowledge representation. There are also relations to abduction and default logic. The work on updating knowledge bases by Guessoum and Lloyd [31, 32], who consider view updates on logic databases, particularly the correctness of updates, is discussed in Chapter 5. Kakas and Mancarella [41] discuss semantics; Kowalski and Sadri [48] consider an exceptions representation for logic programming; Kakas et al [40] review abductive logic programming, for example its relation to truth maintenance.

Semantics of Non-monotonic logic. Konolige [44] has proposed partial models as a semantics for non-monotonic logic; Przyminski [104] considers semantics of non-monotonic formalism in the framework of general logic programs; Nilsson and Genesereth [28] cover many of the essentials of non-monotonic logic in their textbook; Ginsberg [30] has collected many foundational but hard-to-find papers in non-monotonic reasoning (including papers by McCarthy, Lifschitz et al.).

Theory Revision. Gärdenfors [27] has proposed a general theory of updating beliefs.

Defeasible reasoning. Pollock [102] has developed a theory of defeasible reasoning and implemented a prototype system.

8.3 Future directions

We believe that several important open issues may be identified from the foregoing results. These are apart from the attempt to learn a complete of KRK for positions won at depths up to 16. Some suggestions along these lines are given below.

First, we must mention again noise, since although it has not been a theme in the work on KRK, this is not likely to be the case in any real-world applications of our methods. This is particularly true of the specialisation method, where observed exceptions to rules may be due to bad data. Some work on this has

already been completed using techniques based on the theory of algorithmic (Kolmogorov) complexity, and this is reported in [88]. However, this version of non-monotonic learning is not incremental. An incremental approach requires testing theories against large data sets, which involves theorem-proving and is consequently expensive.

Secondly, we have shown that restriction of the hypothesis language, e.g. by restricting vocabulary, is a major obstacle for successful learning. However some vocabulary restriction seems inevitable to achieve efficiency. A possible solution might be to allow a learning agent to conservatively design its vocabulary. Predicate invention does this, as in the CWS algorithm where compact representation of a specialised rule is gained by making a new predicate specific to the exceptions observed. Instances of this new predicate are subsequently generalised. Muggleton has recently proposed a general theoretical framework for predicate invention based on a refinement lattice. Since the CWS approach appears to be a special case within this framework, a clear characterisation of CWS in this way could be very informative.

Finally, a limitation in the current generalisation engine used in GCWS (GOLEM) is that all background predicates must be supplied to the algorithm in the form of ground atoms. This means that although new theories can be induced for the target predicate, and intermediate invented predicates, there is a difficulty with including them in subsequent learning in the same domain or of similar concepts in new domains. The difficulty arises from the ground background restriction in current RLGG implementation GOLEM. This prevents the use of, say, an induced KRK theory as background knowledge in the subsequent induction of KRKN or some other endgame. This points to a topic of great future importance concerned with the utilisation by machine learners of discovered concepts through common predicate libraries. While this thesis has not sought to enter this territory, ground has been cleared for relevant new work in progress elsewhere [84].

8.4 Summary

In this thesis we embarked on the task of learning KRK concepts. We began with some rules of the game and progressed to the induction of rules for winning the endgame. This step was tackled "backwards" from the end, by starting with checkmated positions.

Applying an algorithm developed to implement change of representation enabled successful learning of the main concepts for illegality in the selected domain.

This method of representation change, relying on predicate invention to learn new vocabulary for concept expression was used in the KRK won–optimally concept.

The failure so far of this method to learn concise concepts at depths of win greater than one seems to be due to the restricted predicate invention capacity of this form of representation change. As such, this marks out the limit of the learning techniques developed in the thesis.

Appendix A

Definitions

A.1 Definitions from logic

In this section we give definitions for logical expressions used in this thesis. These are based on those of [18, 56, 82].

A.1.1 Formulae in propositional calculus

A proposition (propositional variable) is represented by a lower case letter followed by a string of lower case letters and digits. The negation symbol is: $\neg$. The binary connectives are: $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. A proposition is an atomic formula, or atom. A compound proposition is an atom, or else it is the negation of a compound proposition, or else it is two compound propositions separated by a binary connective.

A.1.2 Formulae in first order predicate calculus

A variable is represented by an upper case letter followed by a string of lower case letters and digits. A function symbol is a lower case letter followed by a string of lower case letters and digits. A predicate symbol is a lower case letter followed by a string of lower case letter and digits. The negation symbol is: $\neg$. A variable is a term, and a function symbol immediately followed by a

bracketed n-tuple of terms is a term. Thus $f(g(X), h)$ is a term when f, g and h are function symbols and X is a variable. A predicate symbol immediately followed by a bracketted n-tuple of terms is called an atomic formula. Both A and $\neg A$ are literals whenever A is an atomic formula. In this case A is called a positive literal and $\neg A$ is called a negative literal. The literals A and $\neg A$ are said to be each others complements and form, in either order, a complementary pair. A finite set (possibly empty) of literals is called a clause. The empty clause is represented by $\Box$. A clause represents the disjunction of its literals. Thus the clause $\{A_1, A_2, ..\neg A_i, \neg A_{i+1}, ...\}$ can be equivalently represented as $(A_1 \vee A_2 \vee ..\neg A_i \vee \neg A_{i+1} \vee ...)$ or $A_1, A_2, .. \leftarrow A_i, A_{i+1}, ...$ A Horn clause is a clause which contains exactly one positive literal. The positive literal in a Horn clause is called the head of the clause while the negative literals are collectively called the body of the clause. A set of clauses is called a clausal theory. The empty clausal theory is represented by $\blacksquare$. A clausal theory represents the conjunction of its clauses. Thus the clausal theory $\{C_1, C_2, ...\}$ can be equivalently represented as $(C_1 \wedge C_2 \wedge ...)$. A set of Horn clauses is called a logic program. Apart from representing the empty clause and the empty theory, the symbols $\Box$ and $\blacksquare$ represent the logical constants *False* and *True* respectively. Literals, clauses and clausal theories are all well-formed-formulae. Let E be a wff or term. vars(E) denotes the set of variables in E. E is said to be ground if and only if vars(E) = $\emptyset$.

A.2 Definitions from Machine Learning

Terms used in this thesis are defined as follows.

Formalism: The syntax of the hypothesis language. **Hypothesis vocabulary:** Predicate symbols used in constructing hypotheses. **Hypothesis language:** The formalism plus the hypothesis vocabulary. **Background knowledge:** Axiomatisation of the individual symbols in the hypothesis vocabulary. **Instance vocabulary** *or* **piece-on-place attributes:** Position of a chess piece

described in terms of its file (a-h) or rank (1-8). **Instance language:** An instance is a class value paired with a vector of symbols from the instance vocabulary. **Example language:** An example is a class value paired with a vector of symbols from the instance and/or hypothesis vocabulary. **Hypothesis space:** Set of all possible hypotheses within the hypothesis language. **Instance space:** Set of all possible instances in the instance language. **Example space:** Set of all possible examples in the example language.

Relational learning: A relation expresses a relationship between objects. Some examples of relations are: "is less than"; "is an element of"; "are the parents of". The first two examples are binary relations – they have two arguments – while the third is a ternary relation. The general form is an N-ary relation. An N-ary relation is defined extensionally by a set of ordered N-tuples. An N-ary relation is defined intensionally by an N-place predicate.

Appendix B

Proofs and the "deriv" function

B.1 Proof of Theorem 5.7

This appendix contains the lengthy proof of Theorem 5.7 from Section 5.2 in Chapter 5. A related result known as the "Subsumption theorem" was proved by Lee [51]. For the purposes of the proof we will make various definitions based on the theory of resolution theorem proving [115]. We define a derivation expression DE as follows

Either DE is a non-empty clause or DE is the expression $(\langle DE_1, l_1\rangle \cdot \langle DE_2, l_2\rangle)$

where DE_1 and DE_2 are derivation expressions and l_1 and l_2 are literals found in the clauses in DE_1 and DE_2 respectively. The resolvent of a derivation expression is defined recursively as follows.

$$\text{resolvent}(DE) = \begin{cases} DE & \text{if } DE \text{ is a clause} \\ \text{Resolution of } (l_1 \vee E_1)\theta_1 & \text{otherwise if } DE = (\langle DE_1, l_1\rangle \cdot \langle DE_2, l_2\rangle) \\ \quad \text{and } (l_2 \vee E_2)\theta_2 \text{ resolved} & \quad \text{and resolvent}(DE_1) = (l_1 \vee E_1)\theta_1 \\ \quad \text{on } l_1\theta_1 \text{ and } l_2\theta_2 & \quad \text{and resolvent}(DE_2) = (l_2 \vee E_2)\theta_2 \\ & \quad \text{and } l_1\theta_1,\ l_2\theta_2 \text{ is a} \\ & \quad \text{complementary pair} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Since the $dot(\cdot)$ operator represents binary resolution it is commutative, non-associative and non-distributive. These dotted derivation expressions are an extension of notation introduced in [86]. Clearly C is an element of $R^*(T)$ if and only if there exists a derivation expression whose clauses are all elements of T and whose resolvent is C. We say that a derivation expression RE is a refutation expression whenever resolvent$(RE) = \Box$. It follows from this definition of a refutation expression that RE must be a dotted expression and not simply a clause.

Next we define the partial recursive function *deriv: set of unit clauses* $\times$ *derivation expression* $\mapsto$ *derivation expression*. This function transfers the elements of the set of unit clauses found in the given refutation expression to the clause found at the root of a new derivation expression.

$$\text{deriv}(U, DE) = \begin{cases} DE & \text{if } DE \text{ is a clause and } DE \notin U \\ \text{deriv}(U, DE_2) & \text{otherwise if } DE = (\langle \{l_1\}, l_1 \rangle \cdot \langle DE_2, l_2 \rangle) \\ & \qquad \text{where } \{l_1\} \in U \\ \text{deriv}(U, DE_1) & \text{otherwise if } DE = (\langle DE_1, l_1 \rangle \cdot \langle \{l_2\}, l_2 \rangle) \\ & \qquad \text{where } \{l_2\} \in U \\ (\text{deriv}(U, DE_1) \cdot & \text{otherwise if } DE = (\langle DE_1, l_1 \rangle \cdot \langle DE_2, l_2 \rangle) \\ \quad \text{deriv}(U, DE_2)) & \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that deriv(U,DE) is undefined whenever DE is a clause and DE $\in$ U. However, if RE is a refutation expression and $\overline{C}$ is a set of unit clauses then no sub-expression of deriv$(\overline{C}, RE)$ will be undefined. A refutation expression is a derivation expression and thus, by definition, is not an empty clause. Also from the definition of deriv it will be noted that no recursion deriv$(\overline{C}, DE)$ will DE be an element of $\overline{C}$ when C is not a tautology. Clearly there is a one-one correspondence between the sub-expressions of the result of deriv$(\overline{C}, RE)$ and a subset of the sub-expressions of RE.

Theorem 5.7 (Clause entailment using resolution) *Let T be a set of*

clauses and C be a non-tautological clause. $T \vdash C$ if and only if there exists D in $R^*(T)$ and substitution θ such that $D\theta \subseteq C$.

Proof. *The "if" part is trivial since $T \vdash D$ and $D \vdash C$, thus $T \vdash C$. We now prove the "only if" part constructively. Let $C = (l_1 \vee l_2 \vee \ldots)$. Using Lemma 5.5, $T \vdash C$ if and only if $T \wedge \overline{C}$ is unsatisfiable. Let $\overline{C} = (\overline{l_1} \wedge \overline{l_2} \wedge \ldots)\theta_s$ where θ_s is a skolemisation of the variables of C. Note that we can write θ_s^{-1} as a deterministic "deskolemisation" rewrite due to the nature of skolemisation.*

Following a convention introduced in [114], resolution-based refutations and derivations are drawn as binary trees. There is a one-one correspondence between derivation expressions and derivation trees. Figure 38 represents the transformation of a refutation tree into a derivation tree using "deriv". In this figure both the refutation tree and the derivation tree represent general cases. The diagonal dotted lines merely indicate a path formed from any number of resolution steps. A capital letter or its prime with a subscript, such as E_j', stands for a clause. A lower case letter or its prime with a subscript, such as l_i' stands for a literal found within a clause of theory T. A lower case Greek letter or its prime with a subscript, such as θ_k' represents a substitution. Note that sub-trees labelled $F\theta_h$ represent trees of maximum depth $\leq h$. The depth of a tree is defined in terms of its corresponding derivation expression as follows

$$depth(DE) = \begin{cases} 0 & \text{if } DE \text{ is a clause} \\ max(depth(DE_1), depth(DE_2)) + 1 & \text{if } DE = (\langle DE_1, l_1\rangle \cdot \langle DE_2, l_2\rangle) \end{cases}$$

Note also that θ_{i*} nodes in the derivation tree correspond to θ_i nodes in the refutation tree.

Let $D = (l_i' \vee \ldots)\theta_{f*}$. For the purposes of the proof it is necessary to show that there exists a most-general-unifier (mgu) for each resolution step within the derivation tree of Figure 38 and that there exists a substitution θ such that $D\theta \subseteq C$. We will first prove by induction on $k*$ that there exists a substitution σ_{k*} such that $\theta_{k*}\sigma_{k*} = \theta_k$ for all corresponding θ_{k*} and θ_k where $k* \in \{0, 1, .., f*\}$.

For the base case, $k = 0$, and thus the maximum depth of the sub-trees involved in the derivation tree is less than or equal to 0. For any such sub-tree*

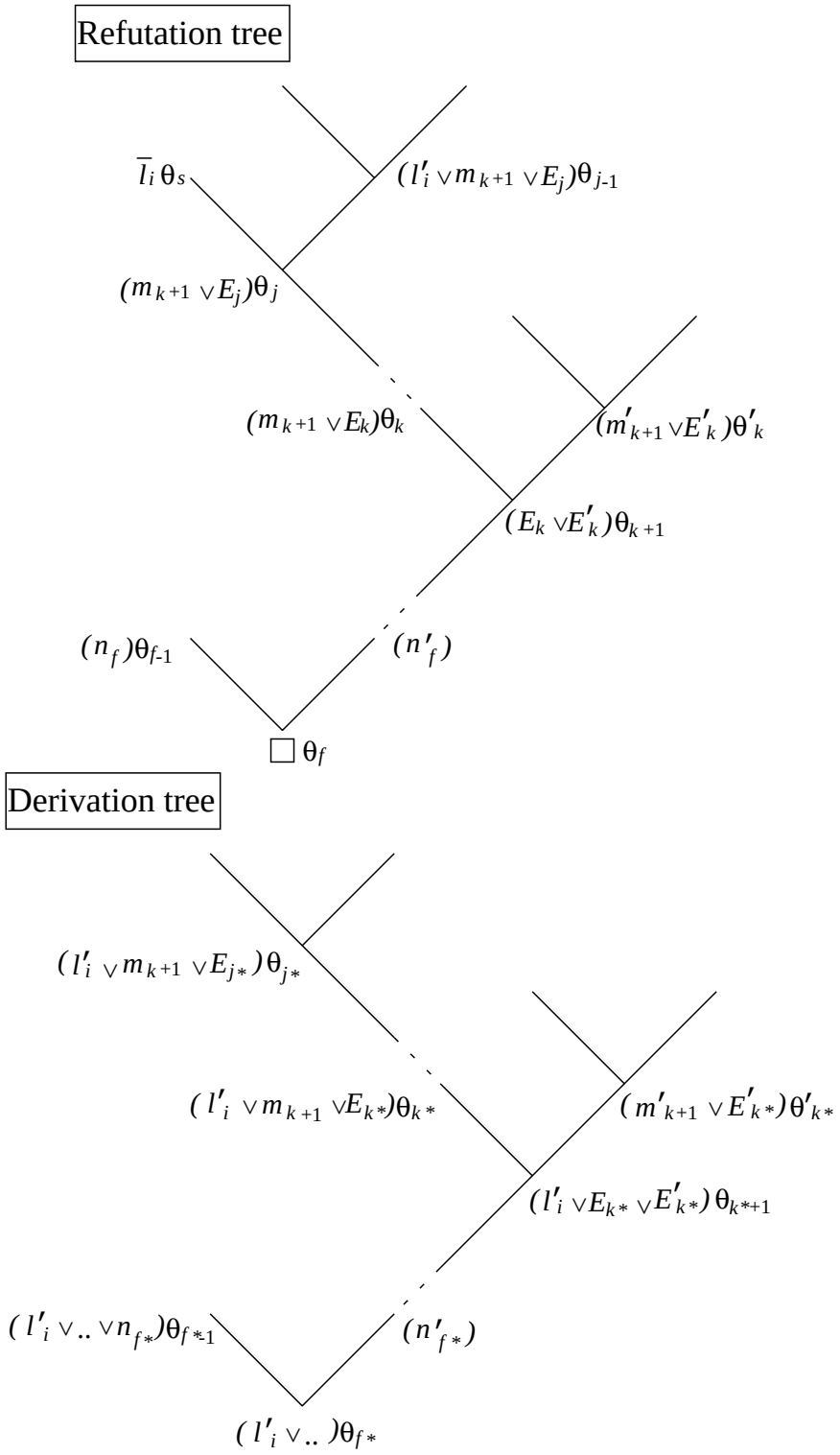


Figure 38: Transforming refutation tree into derivation tree using "deriv".

$\theta_{k*} = \theta_0 = \emptyset$ *and letting* $\sigma_{k*} = \theta_k$, *clearly* $\theta_{k*}\sigma_{k*} = \emptyset\theta_k = \theta_k$ *where* θ_k *is the substitution applied at the corresponding node in the refutation tree. This proves the base case. We make the inductive hypothesis that there exists* σ_{j*} *such that* $\theta_{j*}\sigma_{j*} = \theta_j$ *for all corresponding* θ_{j*} *and* θ_j *where* $0 \leq j* \leq k*$ *and* $k* > 0$. *Now we must show that there exists* σ_{k*+1} *such that* $\theta_{k*+1}\sigma_{k*+1} = \theta_{k+1}$ *for all corresponding* θ_{k*+1} *and* θ_{k+1}. *With reference to Figure 38 and the inductive hypothesis we know that* $m_{k+1}\theta_{k*}\sigma_{k*} = m_{k+1}\theta_k$ *and* $m'_{k+1}\theta'_{k*}\sigma'_{k*} = m'_{k+1}\theta'_k$. *We also know from the refutation tree in Figure 38 that* $m_{k+1}\theta_k$ *and* $m'_{k+1}\theta'_k$ *have an mgu, call it* μ_{k+1}, *such that* $m_{k+1}\theta_k\mu_{k+1} = m'_{k+1}\theta'_k\mu_{k+1}$. *Thus in the derivation tree in Figure 38* $m_{k+1}\theta_{k*}$ *and* $m'_{k+1}\theta'_{k*}$ *must have an mgu, call it* μ_{k*+1}, *since using the inductive hypothesis they at least have the unifier* $\sigma_{k*}\sigma'_{k*}\mu_{k+1}$. *By the definition of an mgu* μ_{k*+1} *is an mgu only if there is some substitution* σ_{k*+1} *such that* $\mu_{k*+1}\sigma_{k*+1} = \sigma_{k*}\sigma'_{k*}\mu_{k+1}$. *Now* $\theta_{k*+1} = \theta_{k*}\theta'_{k*}\mu_{k*+1}$ *and thus* $\theta_{k*+1}\sigma_{k*+1} = \theta_{k*}\theta'_{k*}\mu_{k*+1}\sigma_{k*+1}$. *Also* $\theta_{k+1} = \theta_k\theta'_k\mu_{k+1} = \theta_{k*}\sigma_{k*}\theta'_{k*}\sigma'_{k*}\mu_{k+1}$ *by the inductive hypothesis. Since* θ_{k*} *and* σ_{k*} *share no variables with* θ'_{k*} *and* σ'_{k*}, $\theta_{k+1} = \theta_{k*}\theta'_{k*}\sigma_{k*}\sigma'_{k*}\mu_{k+1}$. *But from above* $\mu_{k*+1}\sigma_{k*+1} = \sigma_{k*}\sigma'_{k*}\mu_{k+1}$ *and thus* $\theta_{k+1} = \theta_{k*}\theta'_{k*}\mu_{k*+1}\sigma_{k*+1} = \theta_{k*+1}\sigma_{k*+1}$, *which proves the inductive step. Thus there is an mgu for every resolution step within the derivation. We must show now that* $D\theta \subseteq C$.

Since $\overline{C} = (\overline{l_1} \wedge \overline{l_2} \wedge \ldots)\theta_s$ *it follows that all* $\overline{l_i}\theta_s$ *are ground literals. Thus if* $(l_i\theta_s, l'_i)$ *is a complementary pair in the refutation tree with mgu* γ_i, *this substitution must be a ground substitution whose domain is the set of variables found in* l'_i. *Also, since* γ_i *is ground,* $\gamma_i \subseteq \theta_f$ *and since* $l'_i\gamma_i = l_i\theta_s$ *it follows that* $(l'_i \vee \ldots)\theta_f \subseteq C\theta_s$. *But we have proved already that* $\theta_f = \theta_{f*}\sigma_{f*}$, *and thus* $(l'_i \vee \ldots)\theta_{f*}\sigma_{f*} \subseteq C\theta_s$. *But* $D = (l'_i \vee \ldots)\theta_{f*}$ *and thus* $D\sigma_{f*} \subseteq C\theta_s$. *Since skolem constants can be unbound without conflict* $D\sigma_{f*}\theta_s^{-1} \subseteq C\theta_s\theta_s^{-1}$, *or* $D\sigma_{f*}\theta_s^{-1} \subseteq C$. *Letting* $\theta = \sigma_{f*}\theta_s^{-1}$ *we complete the proof since this gives* $D\theta \subseteq C$. *QED*

B.2 "Deriv" as an Explanation-Based Generalisation technique

The "deriv" function of Appendix B turns out to be almost entirely isomorphic to the EBG algorithm "prolog_ebg" described in [42]. The main differences between deriv and prolog_ebg are as follows. Firstly, while deriv acts on general clauses, prolog_ebg is restricted to Horn clauses. Secondly, prolog_ebg depends on a user-defined "operationality criterion". No such criterion is necessary for deriv. Thirdly, although EBG is supposed to find a "generalisation of the training example that is a sufficient concept definition for the target concept and that satisfies the operationality criterion" [42], we are not aware of any rigorous definition or proofs concerning these aims. On the other hand, it can be proved for deriv that if there is only one refutation of $T \wedge \overline{C}$ then the resolvent D of the derivation expression returned by deriv is the most general clause which both θ-subsumes C and is entailed by T. This condition places a useful bound on the complexity of D. That is that D must be smaller than C since it θ-subsumes C. No such bound exists for the size of the clause returned by prolog_ebg. In fact, when we ran prolog_ebg on the exponential function coded in Prolog using Peano arithmetic for multiplication and addition the number of literals in the returned clause was exponentially related to the value of the exponent within the example. This seems to indicate that prolog_ebg would not be good at helping to speed up searches in intractable domains such as game playing, though deriv might be.

We have coded a version of deriv called "drv" which runs on Horn clause examples. The Prolog code is given below.

```
% drv(C,D) - finds the most general clause D which is entailed by
%            the background theory and theta-subsumes C. The
%            bodies of C and D are expected to be terminated
%            with the atom ''true''.
```

```
drv((H :- B),(HGen :- BGen)) :-
        functor(H,F,N), functor(HGen,F,N),
        skolemises([(H :- B)],0,_),
        drv(H,B,HGen,BGen,true).

% drv(Goal,Units,GenGoal,GenB,GenBend) - adapted Prolog interpreter
%            which proves the Goal either against program clauses or
%            against the set of skolemised Unit clauses.  D's head (GenGoal)
%            and body (GenB) are formed by carrying out all resolution steps
%            except those involving Units.  The atomic formula ``true'' is
%            passed as GenBend and placed at the end of GenB.

drv(true,_,true,GenB,GenB) :- !.
drv((X,Y),U,(GenX,GenY),GenB,GenBend) :- !,
        drv(X,U,GenX,GenB,NewEnd),
        drv(Y,U,GenY,NewEnd,GenBend).
drv(G,U,GenG,GenB,GenBend) :-
        clause(GenG,GenGs),
        copy_term((GenG :- GenGs),(G :- Gs)),
        drv(Gs,U,GenGs,GenB,GenBend).
drv(Goal,U,GenGoal,(GenGoal,GenB),GenB) :-
        gmem(Goal,U).

% skolemises(Terms,N,M) - binds all variables in Terms to skolem
%            constants in the range [N..M-1]

skolemises([],N,N).
skolemises([$(N)|T],N,M) :- !,
        Np1 is N + 1,
        skolemises(T,Np1,M).
skolemises([Tm|T],N,M) :-
        Tm =..  [_|Tms],
```

```
        skolemises(Tms,N,O),
        skolemises(T,O,M), !.

% gmem(Goal,Units) - unifies Goal to a member of Units if possible.
%            Otherwise fails.

gmem(X,X).
gmem(X,(A,B)) :-
        gmem(X,A) ; gmem(X,B).
```

This was run on the "suicide" example from [42]. The background knowledge was

```
kill(A,B) :- hate(A,B), possess(A,C), weapon(C).
hate(W,W) :- depressed(W).
possess(U,V) :- buy(U,V).
weapon(Z) :- gun(Z).
```

When given the goal

```
drv((kill(john,john) :- depressed(john), buy(john,gun1), gun(gun1),
day(tuesday), true), D)
```

`D` is instantiated to

```
(kill(X,X) :- depressed(X), buy(X,C), gun(C), true).
```

In this case drv produces the same result as prolog_ebg. However, given the standard recursive definition of list membership, the goal

```
drv((member(3, [1,2,3]) :- true), D)
```

instantiates `D` to

```
(member(A,[B,C,A|D]) :- true).
```

On the same problem prolog_ebg merely produces the standard recursive clause

```
(member(A, [B|C]) :- member(A,C))
```

which is not a useful generalisation.

Appendix C

Breakdown of classifier scores

In this appendix we present a breakdown of the individual BTM classifier scores by depth of win. These scores were obtained by testing the separately induced classifiers described in Section 7.3.1 of Chapter 7 under the order-independent control strategy outlined in Section 7.3.2 of Chapter 7.

The key to the presentation of the 2×2 tables in this appendix is as follows:

Classifier Predicts

		"Won"	"Not Won"	
Actual	Won	a	b	a+b
Values	Not Won	d	c	c+d
		a+d	b+c	a+b+c+d

Note that the values Won and Not Won refer to the specific depth-of-win printed above each table.

Depth 0

	"Won"	"Not Won"	
Won	27	0	27
Not Won	99	27930	28029
	126	27930	28056

Depth 1

	"Won"	"Not Won"	
Won	78	0	78
Not Won	490	27488	27978
	568	27488	28056

Depth 2

	"Won"	"Not Won"	
Won	246	0	246
Not Won	238	27572	27810
	484	27572	28056

Depth 3

	"Won"	"Not Won"	
Won	81	0	81
Not Won	739	27236	27975
	820	27236	28056

Depth 4

	"Won"	"Not Won"	
Won	198	0	198
Not Won	1422	26436	27858
	1620	26436	28056

Depth 5

	"Won"	"Not Won"	
Won	471	0	471
Not Won	2961	24624	27985
	3432	24624	28056

Depth 6

	"Won"	"Not Won"	
Won	592	0	592
Not Won	5708	21756	27464
	6300	21756	28056

Depth 7

	"Won"	"Not Won"	
Won	683	0	683
Not Won	8761	18612	27373
	9444	18612	28056

Depth 8

	"Won"	"Not Won"	
Won	1433	0	1433
Not Won	6797	19826	26623
	8230	19826	28056

Depth 9

	"Won"	"Not Won"	
Won	1712	0	1712
Not Won	13554	12790	26344
	15266	12790	28056

Depth 10

	"Won"	"Not Won"	
Won	1985	0	1985
Not Won	15025	11046	26071
	17010	11046	28056

Depth 11

	"Won"	"Not Won"	
Won	2854	0	2854
Not Won	13372	11830	25202
	16226	11830	28056

Depth 12

	"Won"	"Not Won"	
Won	3597	0	3597
Not Won	15451	9008	24459
	19048	9008	28056

Depth 13

	"Won"	"Not Won"	
Won	4194	0	4194
Not Won	14174	9688	23862
	18368	9688	28056

Depth 14

	"Won"	"Not Won"	
Won	4553	0	4553
Not Won	0	23503	23503
	4553	23503	28056

Depth 15

	"Won"	"Not Won"	
Won	2166	0	2166
Not Won	11420	14470	25890
	13586	14470	28056

Depth 16

	"Won"	"Not Won"	
Won	390	0	390
Not Won	2593	25073	27666
	2983	25073	28056

Depth ∞ – draw

	"Won"	"Not Won"	
Won	2796	0	2796
Not Won	0	25260	25260
	2796	25260	28056

Bibliography

[1] M. Altman, A. Cheng, and S. Galeota. AQ11: Experiments and Evaluation. C.S. 397, 1979. University of Illinois at Urbana Champaign.

[2] D. Angluin and C. H. Smith. A survey of inductive inference: theory and methods. *Computing Surveys*, 15(3):237–269, 1983.

[3] M. Bain. Experiments in non-monotonic learning. In L. Birnbaum and G. Collins, editors, *Proceedings of the Eighth International Workshop on Machine Learning*, pages 380–384, San Mateo, CA, 1991. Morgan Kaufmann.

[4] M. Bain. Learning optimal chess strategies. In S. Muggleton, editor, *ILP 92: Proc. Intl. Workshop on Inductive Logic Programming*, volume ICOT TM-1182. Institute for New Generation Computer Technology, Tokyo, Japan, 1992.

[5] M. Bain and S. H. Muggleton. Non-monotonic learning. In J. E. Hayes, D. Michie, and E. Tyugu, editors, *Machine Intelligence 12*, pages 105–119. Oxford University Press, Oxford, 1991.

[6] F. Bergadano and A. Giordana. A knowledge intensive approach to concept induction. In *ML-88: Proc. of the Fifth Intl. Conference on Machine Learning*, pages 305–317, San Mateo, CA, 1988. Morgan Kaufmann.

[7] H. J. Berliner and M. Campbell. Using chunking to play chess pawn endgames. *Artificial Intelligence*, 23:97–120, 1984.

[8] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's Razor. Technical Report UCSC-CRL-86-2, Computer Research Laboratory, University of California at Santa Cruz, 1986.

[9] I. Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, Wokingham, 1990.

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

[11] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.

[12] A. Bundy, B. Silver, and D. Plummer. An analytical comparison of some rule-learning programs. *Artificial Intelligence*, 27:137–181, 1985.

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

[14] W. L. Buntine. *A Theory of Learning Classification Rules*. PhD thesis, University of Sydney, 1992.

[15] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell. An Overview of Machine Learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning*, volume 1, pages 3–23. Tioga, Palo Alto, CA., 1983.

[16] B. Cestnik, I. Kononenko, and I. Bratko. ASSISTANT 86: a knowledge-elicitation tool for sophisticated users. In *Progress in Machine Learning*, pages 31–45, Wilmslow, England, 1987. Sigma.

[17] G. Chaitin. *Information, Randomness and Incompleteness - Papers on Algorithmic Information Theory*. World Scientific Press, Singapore, 1987.

[18] C-L. Chang and R. C-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, London, 1973.

[19] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261–283, 1989.

[20] M. R. B. Clarke. A Quantitative Study of King and Pawn Against King. In M. R. B. Clarke, editor, *Advances in Computer Chess*, volume 1, pages 108–118. Edinburgh University Press, Edinburgh, 1977.

[21] W.F. Clocksin and C.S. Mellish. *Programming in Prolog*. Springer-Verlag, Berlin, 1981.

[22] S. Craw and D. Sleeman. The flexibility of speculative refinement. In L. Birnbaum and G. Collins, editors, *ML-91: Proc. Eighth Intl. Workshop on Machine Learning*, pages 28 – 32, San Mateo, CA, 1991. Morgan Kaufmann.

[23] R. Dechter and D. Michie. Induction of plans. TIRM 84-006, The Turing Institute, Glasgow, 1984.

[24] N. Dershowitz and Y. Lee. Deductive Debugging. UIUCDCS-F- 87-961, University of Illinois at Urbana-Champaign, Urbana, IL, 1987.

[25] C. Feng. Inducing temporal fault diagnostic rules from a qualitative model. In S. H. Muggleton, editor, *Inductive Logic Programming*, pages 473–493. Academic Press, London, 1992.

[26] S. Gallant. Example-based knowledge engineering with conectionist expert systems. In *IEEE Midcon, August 30 - September 1.* Dallas, Texas, 1988.

[27] P. Gardenfors. *Knowledge in Flux: Modeling the Dynamics of Epistemic States.* MIT Press, Cambridge, MA, 1988.

[28] M. R. Genesereth and N. J. Nilsson. *Logical foundations of artificial intelligence.* Morgan Kaufmann, Los Altos, 1987.

[29] A. Ginsberg. Theory Revision via Prior Operationalization. In *AAAI-88: Proc. Seventh National Conf. on Artificial Intelligence*, pages 590 – 595, San Mateo, CA, 1988. Morgan Kaufmann for AAAI.

[30] M. Ginsberg. *Readings in nonmonotonic reasoning.* Morgan Kaufmann, Los Altos, CA, 1987.

[31] A. Guessoum and J. W. Lloyd. Updating Knowledge Bases. *New Generation Computing*, 8(1):71–89, 1990.

[32] A. Guessoum and J. W. Lloyd. Updating Knowledge Bases II. *New Generation Computing*, 10(1):73–100, 1991.

[33] R. Hamakawa. Revision Cost for Theory Refinement. In L. Birnbaum and G. Collins, editors, *ML-91: Proc. Eighth Intl. Workshop on Machine Learning*, pages 514 – 518, San Mateo, CA, 1991. Morgan Kaufmann.

[34] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36:177 – 221, 1988.

[35] N. Helft. Inductive generalisation: a logical framework. In I. Bratko and N. Lavrac, editors, *Progress in Machine Learning (EWSL-87)*, London, 1987. Pitman.

[36] I. S. Herschberg and J. van den Herik. A database on databases. *ICCA Journal*, 8(3):131–139, 1986.

[37] C. A. R. Hoare. Programs are Predicates. In *FGCS-92: Proc. of the Final Conference on New Generation Computer Technology*, Tokyo, 1992. Ohmsha.

[38] E. B. Hunt and C. I. Hovland. Programming a model of concept formation. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 310–328. Krieger, Malabar, FLA., 1963.

[39] K. Jantke. Monotonic and non-monotonic inductive inference. In S. Arikawa et al., editor, *Proceedings of the Workshop on Algorithmic Learning Theory*, pages 269–281, Tokyo, 1990. Ohmsha.

[40] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive logic programming. TM- 1186, ICOT, Tokyo, Japan, 1993.

[41] A. C. Kakas and P. Mancarella. Generalized Stable Models: A Semantics for Abduction. In *ECAI 90: Proceedings of the Ninth European Conference on Artificial Intelligence*, pages 385–391, London, 1990. Pitman.

[42] S.T. Kedar-Cabelli and L.T. McCarty. Explanation-based generalization as resolution theorem proving. In P. Langley, editor, *Proceedings of the Fourth International Workshop on Machine Learning*, pages 383–389, Los Altos, 1987. Morgan Kaufmann.

[43] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Prob. Inf. Trans.*, 1:1–7, 1965.

[44] K. Konolige. Partial models and non-monotonic inference. In J.E. Hayes, D. Michie, and J. Richard, editors, *Machine Intelligence 11*, pages 3–19. Oxford University Press, Oxford, 1988.

[45] I. Kononenko. ID3, Sequential Bayes, Naive Bayes and Bayesian Neural Networks. In *Proceedings of the Fourth European Working Session on Learning*, pages 91–98. Pitman, 1989.

[46] I. Kononenko. Interpretation of neural networks decisions. In *Proceedings of IASTED Internat. Conf. on Expert Systems and Applications*. Zurich, 1989.

[47] D. Kopec and T. Niblett. How hard is the play of the KRKN ending ? In M. R. B. Clarke, editor, *Advances in Computer Chess*, volume 2, pages 57–82. Edinburgh University Press, Edinburgh, 1980.

[48] R. A. Kowlaski and F. Sadri. Logic programs with exceptions. In D. H. D. Warren and P. Szeredi, editors, *Logic Programming: Proceedings of the Seventh International Conference*, pages 598–613, Cambridge, MA, 1990. MIT Press.

[49] J.-L. Lassez and K. Marriott. Explicit Representation of Terms Defined by Counter Examples. *Journal of Automated Reasoning*, 3:301–317, 1987.

[50] N. Lavrac and S. Dzeroski. Learning of relational descriptions from noisy examples. In S. H. Muggleton, editor, *Inductive Logic Programming*, Wokingham, UK, 1991. Turing Institute Press with Addison Wesley.

[51] C. Lee. *A completeness theorem and a computer program for finding theorems derivable from given axioms.* PhD thesis, University of California, Berkeley, 1967.

[52] K. Lindsay, B. Buchanan, E. Feigenbaum, and J. Lederberg. DENDRAL: a case study of the first expert system for scientific hypothesis formation. *Artificial Intelligence*, 61(2):209–261, 1993.

[53] X. C. Ling. Inductive Learning from Good Examples. In *IJCAI-87: Proc. Twelfth Intl. Joint Conf. on Artificial Intelligence*, pages 751 – 756, Los Altos, CA, 1991. Morgan Kaufmann.

[54] X. C. Ling. Non-monotonic Specialization. Unpublished, 1991. First International Workshop on Inductive Logic Programming, Viana de Castelo, Portugal.

[55] X. C. Ling and M. Valtorta. Revision of Reduced Theories. In L. Birnbaum and G. Collins, editors, *ML-91: Proc. Eighth Intl. Workshop on Machine Learning*, pages 519 – 523, San Mateo, CA, 1991. Morgan Kaufmann.

[56] J. W. Lloyd. *Logic Programming.* Springer-Verlag, Berlin, 1984.

[57] J. W. Lloyd. *Logic Programming, 2nd Edition*. Springer-Verlag, Berlin, 1987.

[58] W. Marek and M. Truszczynski. Autoepistemic Logic. *Journal of the ACM*, 38(3):588–619, 1991.

[59] J. McCarthy. Applications of circumscription to formalizing common sense knowledge. *Artificial Intelligence*, 28:89–116, 1986.

[60] D. McDermott. A critique of pure reason (Article plus commentaries). *Computational Intelligence*, 3(3):151–237, August 1987.

[61] R. Michalski and J. Larson. Selection of most representative training examples and incremental generation of VL1 hypotheses: the underlying methodology and the description of programs ESEL and AQ11. UIUCDCS-R 78-867, Computer Science Department, Univ. of Illinois at Urbana-Champaign, 1978.

[62] R. Michalski and J. Larson. Incremental generation of vl1 hypotheses: the underlying methodology and the description of program AQ11. ISG 83-5, Computer Science Department, Univ. of Illinois at Urbana-Champaign, 1980.

[63] R. S. Michalski. A theory and methodology of inductive learning. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 83–134. Tioga, Palo Alto, CA, 1983.

[64] R. S. Michalski and P. Negri. An experiment in inductive learning in chess end-games. In E. W. Elcock and D. Michie, editors, *Machine Intelligence*, volume 8, pages 168–201. Edinburgh University Press, Edinburgh, 1977.

[65] D. Michie. King and Rook Against King: Historical Background and a Problem on the Infinite Board. In M. R. B. Clarke, editor, *Advances*

in Computer Chess, volume 1, pages 30–59. Edinburgh University Press, Edinburgh, 1977.

[66] D. Michie. Computer Chess and the Humanisation of Technology. In D. Michie, editor, *On Machine Intelligence (Second Edition)*, pages 77 – 86. Ellis Horwood, Chichester, 1986.

[67] D. Michie. The superarticulacy phenomenon in the context of software manufacture. *Proceedings of the Royal Society of London*, A 405:185–212, 1986.

[68] D. Michie. Towards a knowledge accelerator. In D. F. Beal, editor, *Advances in Computer Chess*, volume 4, pages 1–8. Pergamon Press, Oxford, 1986.

[69] D. Michie. Current developments in expert systems. In J. R. Quinlan, editor, *Applications of expert systems*, pages 137–156. Turing Institute Press in association with Addison-Wesley, Wokingham, 1987.

[70] D. Michie. Machine learning in the next five years. In *Proceedings of the Third European Working Session on Learning*, pages 107–122. Pitman, 1988.

[71] D. Michie. Problems of computer-aided concept formation. In J. R. Quinlan, editor, *Applications of expert systems (Vol. 2)*, pages 310–333. Turing Institute Press in association with Addison-Wesley, Wokingham, 1989.

[72] D. Michie. Personal models of rationality. *Journal of Statistical Planning and Inference*, 25:381–399, 1990.

[73] D. Michie and M. Bain. Machine acquisition of concepts from sample data. In D. Kopec and R. Thompson, editors, *Artificial intelligence and intelligent tutoring systems: knowledge-based systems for teaching and learning*, pages 5–23. Ellis Horwood, Chichester, 1992.

[74] D. Michie and D. Kopec. Mismatch between machine representations and human concepts: dangers and remedies. FAST Series No. 9, 1983. Brussels: Commission of the European Communities.

[75] T.M. Mitchell. Generalisation as search. *Artificial Intelligence*, 18:203–226, 1982.

[76] R. Mooney and B. Richards. Automated Debugging of Logic Programs via Theory Revision. In S. Muggleton, editor, *Proceedings of the Second International Workshop on Inductive Logic Programming*, Institute for New Generation Computer Technology, Tokyo, Japan, 1992. ICOT TM-1182.

[77] S. Muggleton. Inverting Implication. In *ILP-92: Proc. of the Second Intl. Workshop on Inductive Logic Programming*, Tokyo, 1992. ICOT TM 1182 - Institute for New Generation Computer Technology.

[78] S. Muggleton. Predicate invention and utility. *Journal of Experimental and Theoretical Artificial Intelligence*, (to appear), 1993.

[79] S. Muggleton, A. Srinivasan, and M. Bain. Compression, Significance and Accuracy. In D. Sleeman and P. Edwards, editors, *ML-92: Proc. of the Ninth Intl. Workshop on Machine Learning*, pages 338–347, San Mateo, CA, 1992. Morgan Kaufmann.

[80] S. H. Muggleton. Duce, an oracle-based approach to constructive induction. In *IJCAI-87*, pages 287–292, Los Altos, CA, 1987. Kaufmann.

[81] S. H. Muggleton. A strategy for constructing new predicates in first order logic. In *Proceedings of the Third European Working Session on Learning*, pages 123–130. Pitman, 1988.

[82] S. H. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8:295–318, 1991.

[83] S. H. Muggleton. Inverting the resolution principle. In J. E. Hayes, D. Michie, and E. Tyugu, editors, *Machine Intelligence 12*, pages 93–103, Oxford, 1991. Oxford University Press.

[84] S. H. Muggleton, 1993. Personal Communication.

[85] S. H. Muggleton, M. E. Bain, J. Hayes-Michie, and D. Michie. An experimental comparison of human and machine learning formalisms. In A. Segre, editor, *Proceedings of the Sixth International Workshop on Machine Learning*, pages 113–118, Los Altos, CA, 1989. Kaufmann.

[86] 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.

[87] S. H. Muggleton and C. Feng. Efficient induction of logic programs. In S. Arikawa et al., editor, *Proceedings of the Workshop on Algorithmic Learning Theory*, pages 368–381, Tokyo, 1990. Ohmsha.

[88] S. H. Muggleton, A. Srinivasan, and M. Bain. MDL codes for non-monotonic learning. Technical report, Turing Institute, Glasgow, 1991.

[89] A. Newell, J. Shaw, and H. Simon. Chess-Playing Programs and the Problem of Complexity. In D. Levy, editor, *Computer Games 1*, pages 89 – 115. Springer, New York, 1988.

[90] T. Niblett. A study of generalisation in logic programs. In *EWSL-88*, London, 1988. Pitman.

[91] P. O'Rorke. A comparative study of inductive learning systems AQ11P and ID3 using a chess end-game test problem. ISG 82-2, Computer Science Department, Univ. of Illinois at Urbana-Champaign, 1982.

[92] D. Ourston and R. Mooney. Changing the rules: a comprehensive approach to theory refinement. In *AAAI-90: Proc. of the Eighth National*

Conference on Artificial Intelligence, pages 815–820, Menlo Park, CA, 1990. AAAI Press/The MIT Press.

[93] D. Ourston and R. Mooney. Constructive Induction in Theory Refinement. In L. Birnbaum and G. Collins, editors, *ML-91: Proc. of the Eighth Intl. Workshop on Machine Learning*, pages 178–182, San Mateo, CA, 1991. Morgan Kaufmann.

[94] D. Ourston and R. Mooney. Improving Shared Rules in Multiple Category Domain Theories. In L. Birnbaum and G. Collins, editors, *ML-91: Proc. of the Eighth Intl. Workshop on Machine Learning*, pages 534–538, San Mateo, CA, 1991. Morgan Kaufmann.

[95] G. Pagallo. Learning DNF by decision trees. In *IJCAI-89: Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 639–644, Los Altos, CA, 1989. Morgan Kaufmann.

[96] A. Paterson. An attempt to use cluster to synthesise humanly intelligible subproblems for the kpk chess endgame. Technical Report UIUCDCS-R-83-1156, Univ. Illinois, Urbana, IL, 1983.

[97] M. Pazzani, C. Brunk, and G. Silverstein. A knowledge-intensive approach to learning relational concepts. In *ML-91: Proc. of the Eighth Intl. Workshop on Machine Learning*, pages 432–436, San Mateo, CA, 1991. Morgan Kaufmann.

[98] D. Pearce. The induction of fault diagnosis systems from qualitative models. In *AAAI-88: Proceedings of the 7th National Conference on Artificial Intelligence*, pages 353–357, San Mateo, CA, 1988. Morgan Kaufmann.

[99] G. D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, Edinburgh, 1969.

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

[101] G. D. Plotkin. A further note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 6, pages 101–124. Edinburgh University Press, Edinburgh, 1971.

[102] J. L. Pollock. How to reason defeasibly. *Artificial Intelligence*, 57:1–42, 1992.

[103] R. J. Popplestone. An experiment in automatic induction. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 203–215. Edinburgh University Press, Edinburgh, 1969.

[104] T. Przymusinski. Three-valued nonmonotonic formalisms and semantics of logic programs. *Artificial Intelligence*, 49:309–343, 1991.

[105] J. R. Quinlan. Semi-autonomous acquisition of pattern-based knowledge. In D. Michie, editor, *Introductory readings in expert systems*, pages 192–207. Gordon and Breach, New York, 1982.

[106] J. R. Quinlan. Learning efficient classification procedures and their application to chess end games. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 464–482. Tioga, Palo Alto, CA, 1983.

[107] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[108] J. R. Quinlan. Generating production rules from decision trees. In *Proceedings of the Tenth International Conference on Artificial Intelligence*, pages 304–307, Los Altos, CA:, 1987. Kaufmann.

[109] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239–266, 1990.

[110] J. R. Quinlan, P. J. Compton, K.A. Horn, and L. Lazarus. Inductive knowledge acquisition: A case study. In *Proceedings of the Second Australian Conference on the Applications of Expert Systems*, pages 183–204, Sydney, 1986. New South Wales Institute of Technology.

[111] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and databases*. Plenum, New York, 1978.

[112] J. C. Reynolds. Transformational Systems and the Algebraic Structure of Atomic Formulas. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 135–151. Edinburgh University Press, Edinburgh, 1969.

[113] B. Richards and R. Mooney. First-Order Theory Revision. In L. Birnbaum and G. Collins, editors, *ML-91: Proc. of the Eighth Intl. Workshop on Machine Learning*, pages 447–451, San Mateo, CA, 1991. Morgan Kaufmann.

[114] J. A. Robinson. Automatic deduction with hyper-resolution. *International Jounal of Computer Mathematics*, 1:227–234, 1965.

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

[116] C. Rouveirol and J-F Puget. A simple solution for inverting resolution. In *EWSL'89: Proceedings of the Fourth European Working Session on Learning, Montpellier, 1989*, pages 201–210, London, 1989. Pitman.

[117] A. J. Roycroft. The GBR code: a 4-digit code for precise specification of chess force. *Journal of the International Computer Chess Association*, 7(1):53–54, 1984.

[118] A. J. Roycroft. Database "oracles": Necessary and desirable features. *ICCA Journal*, 8(2):100–104, 1986.

[119] 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.

[120] J. C. Schlimmer and D. H. Fisher. A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 496–501, Philadelphia, PA:, 1986. Kaufmann.

[121] C. Shannon. Programming a computer to play chess. *Philosophical Magazine*, 8(3):131–139, 1950.

[122] A. Shapiro and T. Niblett. Automatic induction of classification rules for a chess end-game. In M. R. B. Clarke, editor, *Advances in Computer Chess*, volume 3, pages 73–92. Pergamon Press, Oxford, 1982.

[123] A. D. Shapiro. *Structured Induction in Expert Systems.* Turing Institute Press with Addison Wesley, Wokingham, UK, 1987.

[124] A. D. Shapiro and D. Michie. A self-commenting facility for inductively synthesised endgame expertise. In D.F. Beal, editor, *Advances in Computer Chess*, volume 4, pages 147–165. Pergammon, Oxford, 1986.

[125] E. Y. Shapiro. Inductive inference of theories from facts. Technical Report 192, Dept. Comp. Sci., Yale University, Connecticut, 1981.

[126] E. Y. Shapiro. *Algorithmic Program Debugging.* MIT Press, Cambridge, MA., 1983.

[127] E.H. Shortliffe and B. Buchanan. A model of inexact reasoning in medicine. *Mathematical Biosciences*, 23:351–379, 1975.

[128] H.A. Simon. Why should machines learn? In R. Michalski, J. Carbonnel, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 25–37. Tioga, Palo Alto, CA, 1983.

[129] R.J. Solomonoff. A formal theory of inductive inference. *J. Comput. Sys.*, 7:376–388, 1964.

[130] S. T. Tan. Describing pawn structures. In M. R. B. Clarke, editor, *Advances in Computer Chess*, volume 1, pages 74–88. Edinburgh University Press, Edinburgh, 1977.

[131] S. Tangkitvanich and M. Shimura. Refining a Relational Theory with Multiple Faults in the Concept and Sub-concepts. In *ML-92: Proc. of the Ninth Intl. Conference on Machine Learning*, pages 436–444, San Mateo, CA, 1991. Morgan Kaufmann.

[132] K. Thompson. Retrograde Analysis of Certain Endgames. *ICCA Journal*, 8(3):131–139, 1986.

[133] P.E. Utgoff. Adjusting bias in concept learning. In *IJCAI-83*, pages 447–449, Los Angeles, CA, 1983. Kaufmann.

[134] J. van den Herik and I. S. Herschberg. The construction of an omniscient endgame database. *ICCA Journal*, 8(2):66–87, 1985.

[135] M. H. van Emden and R. A. Kowalski. The Semantics of Predicate Logic as a Programming Language. *JACM*, 23(4):733–742, 1976.

[136] A. van Gelder, K.A. Ross, and J.S. Schlipf. The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.

[137] S. A. Vere. Multilevel Counterfactuals for Generalizations of Relational Concepts and Productions. *Artificial Intelligence*, 14:139–164, 1980.

[138] R. G. Wade. *The Batsford Book of Chess*. B. T. Batsford, London, UK, 1984.

[139] R. Wirth. Completing logic programs by inverse resolution. In *EWSL-89*, pages 239–250, London, 1989. Pitman.

[140] J. Wogulis. Revising Relational Domain Theories. In L. Birnbaum and G. Collins, editors, *ML-91: Proc. of the Eighth Intl. Workshop on Machine Learning*, pages 462–466, San Mateo, CA, 1991. Morgan Kaufmann.

[141] S. Wrobel. Automatic representation adjustment in an observational discovery system. In *EWSL-88*, pages 253–262, London, 1988. Pitman.

[142] S. Wrobel. On the proper definition of minimality in specialization and theory revision. In *ECML-93: Proc. of the European Conf. on Machine Learning*, 1993.