Imperial College London
Department of Computing

Efficient instance and hypothesis space revision in Meta-Interpretive Learning

Céline Hocquette

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College London and
the Diploma of Imperial College London, March 2022

Abstract

Inductive Logic Programming (ILP) is a form of Machine Learning. The goal of ILP is to induce hypotheses, as logic programs, that generalise training examples. ILP is characterised by a high expressivity, generalisation ability and interpretability. Meta-Interpretive Learning (MIL) is a state-of-the-art sub-field of ILP. However, current MIL approaches have limited efficiency: the sample and learning complexity respectively are polynomial and exponential in the number of clauses. My thesis is that improvements over the sample and learning complexity can be achieved in MIL through instance and hypothesis space revision. Specifically, we investigate 1) methods that revise the instance space, 2) methods that revise the hypothesis space and 3) methods that revise both the instance and the hypothesis spaces for achieving more efficient MIL.

First, we introduce a method for building training sets with active learning in Bayesian MIL. Instances are selected maximising the entropy. We demonstrate this method can reduce the sample complexity and supports efficient learning of agent strategies. Second, we introduce a new method for revising the MIL hypothesis space with predicate invention. Our method generates predicates bottom-up from the background knowledge related to the training examples. We demonstrate this method is complete and can reduce the learning and sample complexity. Finally, we introduce a new MIL system called *MIGO* for learning optimal two-player game strategies. *MIGO* learns from playing: its training sets are built from the sequence of actions it chooses. Moreover, *MIGO* revises its hypothesis space with Dependent Learning: it first solves simpler tasks and can reuse any learned solution for solving more complex tasks. We demonstrate *MIGO* significantly outperforms both classical and deep reinforcement learning. The methods presented in this thesis open exciting perspectives for efficiently learning theories with MIL in a wide range of applications including robotics, modelling of agent strategies and game playing.

Acknowledgements

I am extremely grateful to my supervisor Stephen Muggleton for his dedicated involvement and guidance. I sincerely thank him for his invaluable advice and continuous support. I especially am thankful to him for sharing his passion for research and science and immense knowledge throughout the most inspiring discussions.

I would like to thank my examiners Alessandra Russo and Hendrik Blockeel for their valuable feedbacks. I also am grateful to my second supervisor Krysia Broda for her helpful and insightful advice. I also am thankful to Katsumi Inoue to have welcomed me at the National Institute of Informatics. I would also like to thank Lun Ai, Stassa Patsantzis and Wang-Zhou Dai for the friendly and enriching work atmosphere. My appreciation also goes to Zaichen, Patrick and Antoine for their encouragement and lasting friendship.

I am deeply grateful to my parents, my sisters, Aurélie and Élise, and to Léo and Thaïs for their unwavering support and encouragement. Finally, I would like to thank Tony for his cheerful company.

Declaration of originality

I hereby declare that all work presented in this thesis is my own work except where stated otherwise by appropriate reference.

Copyright

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Objectives

1.1.1 Motivation

Induction is the process of inferring general rules from specific observations. Induction is the primary task of machine learning on which we will focus in this thesis. However, induction is inherently difficult due to large search spaces [Rendell, 1985].

ILP [Muggleton, 1991] is a form of Machine Learning. An ILP learner takes as input examples and background knowledge and outputs an hypothesis which, together with the background knowledge, explains and generalises the examples. In ILP, the examples, the background knowledge and learned hypotheses are represented as logic programs. ILP systems have several benefits compared to other forms of Machine Learning [Cropper *et al.*, 2020a]. First, the logical representation is highly expressive which allows to learn complex hypotheses involving recursions, first-order and higher-order logic. ILP systems also have the ability to perform high-level reasoning. Expressivity together with high-level reasoning facilitate high generalisation. Second, ILP systems benefit from a strong inductive bias and can easily make use of prior knowledge. They can reuse and recompose knowledge to build new knowledge which promotes data efficiency. Moreover, since learned hypotheses are represented in the same form as the background

knowledge, they can easily be explicitly added to the background knowledge which provides a natural support for transfer learning and lifelong learning. Finally, the logical representation is humanly comprehensible: learned models have the potential to be read and understood by humans which is crucial for explainable AI. We are interested in this thesis in inducing programs from data with ILP. However, the search for programs is a hard combinatorial problem: the number of programs in any non-trivial programming language grows exponentially with program size.

Meta-Interpretive Learning (MIL) [Muggleton *et al.*, 2014; Muggleton *et al.*, 2015] is a state-of-the-art ILP framework. As a form of ILP, a MIL learner takes as input a set of examples and background knowledge represented as logic programs. A distinctive feature of MIL is the use of meta-rules as part of the background knowledge. Meta-rules are higher-order clauses which act as program templates and specify the form of learned programs allowed in the hypothesis space. The major strengths of MIL is that it supports predicate invention, learning of recursive programs and learning of higher-order programs. However, current MIL systems have limited efficiency. We will investigate in this thesis improvements over MIL efficiency. We measure efficiency of learning systems along two axes: the sample complexity evaluates the number of examples required to converge toward accurate hypotheses while the learning complexity refers to the computational resources required to converge toward accurate hypotheses. Current MIL approaches have restricted efficiency: they have a sample complexity polynomial in the size of learned programs and a learning complexity exponential in the size of learned programs [Muggleton *et al.*, 2015], where the size of a program is measured as its number of clauses.

We introduce in this thesis revision methods for more efficient MIL. First, we investigate methods that revise the instance space. The instance space is the set of items over which hypotheses are defined. The training set is the set of examples presented to the learner. Training examples are labelled instances sampled from the instance space. We investigate methods to guide the selection of informative examples. Second, we investigate methods to revise the hypothesis space. The hypothesis space is the set of all hypotheses that are constructable and may be output by the learner. The hypothesis space is defined by the hypothesis language. We investigate methods which revise the hypothesis space to guide the search for consistent hypotheses.

Third, we investigate methods that revise both the instance and the hypothesis space to achieve more efficient MIL.

We restrict the scope of this thesis to monotonic learning within the Herbrand semantics and learn hypotheses which are definite programs. In Chapter 6, we extend the class of learnable programs and learn programs with stratified negation, in the monotonic learning setting.

1.1.2 Thesis Statement

We claim that more efficient MIL can be achieved. Specifically, my thesis is that improvements over the sample complexity and the learning complexity can be achieved in MIL through:

Subthesis S.1 methods that revise the instance space,

Subthesis S.2 methods that revise the hypothesis space and

Subthesis S.3 methods that revise both the instance space and the hypothesis space.

To support this claim, we investigate whether at least one such method exists in each of the cases above.

1.1.3 Revising the instance space

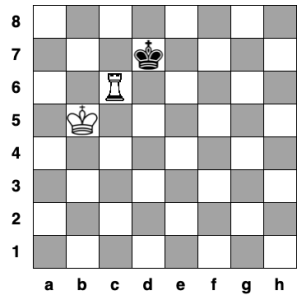


Figure 1.1: Learning a chess pattern: board example of a situation in which the white king protects its rook. It is black-to-move.

An optimal strategy in the chess endgame KRK (King-and-Rook versus King) is to successively restrict the area available to the opponent's black king using the white rook [Bratko, 1978].

Maintenance of white rook safety must be ensured throughout the endgame and this can be achieved using the white king. This concept of rook protection is a crucial feature of a winning strategy in the KRK endgame. Figure 1.1 represents an example of a situation in which the white king protects its rook from the black king. Suppose a learner aims to learn the concept of rook protection represented by the predicate *rp/1*. Suppose also the learner is given as training set a positive example, which is the following atom representing the situation in Figure 1.1:

$$rp([c(5, b, \text{white}, \text{king}), c(6, c, \text{white}, \text{rook}), c(7, d, \text{black}, \text{king})])$$

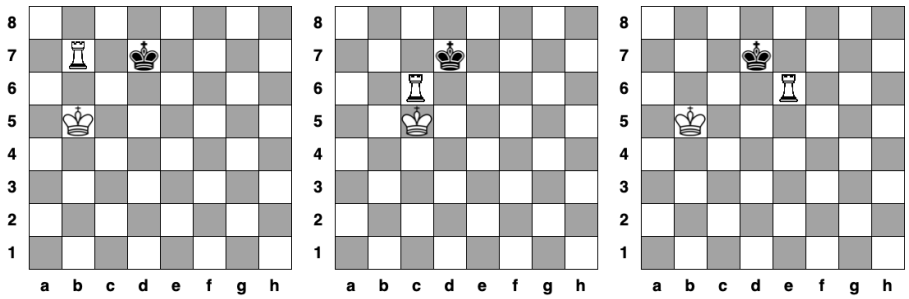
H_1	`rp(A):-rp_1(A,B),rp_2(A,B).`
	`rp_1(A,B):-piece(A,B),white(B).`
	`rp_2(A,B):-rp_1(A,C),distance1(C,B).`
		Board A contains two white pieces at distance 1 of each other.

H_2	`rp(A):-piece(A,B),rp_1(A,B).`
	`rp_1(A,B):-piece(A,C),rp_2(C,B).`
	`rp_2(A,B):-distance1(A,B),black(B).`
		Board A contains a piece at distance 1 of a black piece.

H_3	`rp(A):-piece(A,B),rp_1(A,B).`
	`rp_1(A,B):-piece(A,C),rp_2(C,B).`
	`rp_2(A,B):-distance1(A,B),rp_3(B).`
	`rp_3(A):-white(A),king(A).`
		Board A contains a piece at distance 1 of a white king.

H_4	`rp(A):-rp_1(A,B),rp_2(A,B).`
	`rp_1(A,B):-piece(A,B),white(B).`
	`rp_2(A,B):-piece(A,B),king(B).`
		Board A contains a white king.

Table 1.1: Learning a chess pattern: competing hypotheses



(a) Query 1 (b) Query 2 (c) Query 3

Figure 1.2: Learning a chess pattern: candidate queries. It is black-to-move for each of them.

Given the background predicates *piece/2*, *white/1*, *black/1*, *rook/1*, *king/1* and *distance1/2*, the learner could formulate several hypotheses which generalise this example. Among them, there are the competing hypotheses H_1, H_2, H_3 and H_4. These hypotheses are represented

in Table 1.1 as logic programs along with an English description. All these hypotheses are consistent with the training set and the learner should attempt to discriminate between them. We additionally assume the learner is provided with the set of additional unlabelled instances represented in Figure 1.2 and that the learner is allowed to request the label of any of them to an oracle. Once queried, an instance is labelled and becomes an example which is added to the training set. One can notice Query 1 is consistent with one hypothesis, Query 2 with all four hypotheses and Query 3 with two hypotheses as shown in Table 1.2. Then, all hypotheses having equal prior probability, the instance whose classification should be requested is Query 3. Query 3 is the most informative instance because regardless of its classification, the knowledge of its label will allow the learner to reject exactly half of the current hypotheses. Conversely, Query 1 and Query 2 have a smaller expected proportion of hypotheses rejected. More generally, the most informative instances are those that can be classified the least reliably by the current competing hypotheses. The selection of these informative instances results in a faster reduction of the set of consistent hypotheses. Therefore, when following this query strategy, the learner needs to request fewer labels before converging, which is worthwhile when labelling instances has an associated cost.

	Query 1	Query 2	Query 3
H_1	✗	✓	✗
H_2	✗	✓	✓
H_3	✗	✓	✗
H_4	✓	✓	✓

Table 1.2: Consistency of competing hypotheses with respect to the candidate queries

In Chapter 4, we present an instance selection method for revising the instance space based on this idea. We demonstrate this method helps to converge faster toward accurate hypotheses and to reduce the overall cost of labelling instances. These results support Subthesis S.1.

1.1.4 Revising the hypothesis space

An accurate hypothesis for learning the concept of rook protection is represented in Table 1.3. This hypothesis includes various invented predicates representing sub-concepts. For instance,

rp_1/2 defines the existence of a white king in a board. The predicates involved in this hypothesis can be divided between surface and substrate as in Table 1.3. The substrate is a set of lower level invented predicates and the surface is a set of higher level predicates built from these substrate predicates. Substrate predicates are generated in a first step. Their generation makes available a number of intermediate concepts reusable during the construction of the surface hypothesis. The surface hypothesis has fewer clauses owing to the use of substrate predicates and therefore it is easier to learn. To that extent the generation of substrate predicates changes the learning bias, thus revises the hypothesis space for more efficient learning.

In Chapter 5, we introduce a method for partially delegating the construction of invented predicates. The substrate invented predicates are generated in a first step from the background knowledge by a bottom-up learner. These invented predicates are added to the background knowledge. The surface hypothesis is subsequently built by a top-down learner which can reuse these invented predicates. We theoretically demonstrate our bottom-up predicate construction method is complete. We also demonstrate it reduces the number of clauses to be learned in the surface hypothesis, which can reduce the sample complexity and improve learning performance. These results support Subthesis S.2.

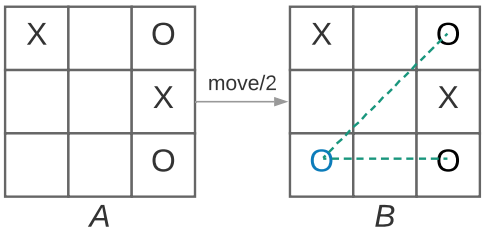
1.1.5 Revising both the instance space and the hypothesis space

An optimal strategy for winning in two moves at Noughts-and-Crosses is to lead double attacks when possible. An example of such double attack is represented in Figure 1.3a. Player O executes a move from board A to board B which creates the two threats represented with the green dotted lines. This results in a forced win for O. More generally, the logic program

Surface	`rp(A):-rp_1(A,B),rp_3(A,B).`

Substrate	`rp_1(A,B):-rp_2(A,B),white(B).`
	`rp_2(A,B):-piece(A,B),king(B).`
	`rp_3(A,B):-rp_4(A,C),distance1(C,B).`
	`rp_4(A,B):-piece(A,B),rook(B).`

Table 1.3: Learning a chess pattern: Target Hypothesis



(a) Example of optimal move: O creates a double attack

```
win_2(A,B):-win_2_1_1(A,B),not(win_2_1_1(B,C)).
  win_2_1_1(A,B):-move(A,B),not(win_1(B,C)).
       win_1(A,B):- move(A,B),won(B).
```

(b) Logic program describing an optimal strategy for Noughts-and-Crosses

Figure 1.3: Optimal Strategy for winning in two moves at Noughts-and-Crosses: O makes a move such that X cannot immediately win nor make a move that blocks O.

represented in Figure 1.3b describes an optimal strategy for winning in two moves at Noughts-and-Crosses. A, B and C are variables which represent state descriptions and encode the current board together with the current player. This strategy states that a move by the current player from state A to state B is a winning move if the opponent cannot immediately win in one move from state B and if the opponent cannot make a move from state B to state C to prevent an immediate win in one move by the current player. This strategy has a hierarchical structure and is decomposed in multiple predicates. For instance, it uses the predicate *win_ 1/2* which represents a strategy for winning in one move. In this way, the learner decomposes a complex problem, such as winning in two moves, into related sub-problems, such as winning in one move. It solves these related sub-problems and saves any solution found. Then, it can reuse any of the simpler concepts learned to learn more efficiently the most complex ones. The ability to solve related problems and remember their solutions shapes the hypothesis space, thus guides the search and facilitates learning.

Moreover, the learner needs to navigate through the large space of possible games to obtain examples. When learning game strategies, the learner can learn from playing to obtain informative examples. In this case, it executes its current strategy: the current strategy guides the choice of examples and is used to build training sets. In other words, the learner constantly is testing its current strategy in order to improve it.

In Chapter 6, we introduce a new learning system called *MIGO* for learning optimal two-player game strategies. *MIGO* orders and solves tasks by increasing complexity. The complexity is measured as the number of moves until completion of the game. Any solution learned is saved in the background knowledge such that it can be reused for solving increasingly more complex tasks. Thus, the hypothesis space is revised as more tasks are solved. Moreover, *MIGO* builds training sets from playing. The examples are generated from the execution of the current strategy and are the result of the actions chosen by *MIGO*. *MIGO* learns optimal strategies for evaluable games in which Minimax Regret can be efficiently evaluated. This allows to have a measure for evaluating learning performance. We experimentally demonstrate *MIGO* significantly outperforms both standard and deep reinforcement learning in terms of Cumulative Minimax Regret when learning evaluable games such as Noughts-and-Crosses. In addition, *MIGO* can achieve significant transfer learning between different domains. Finally, learned strategies are shown to be relatively easy to comprehend. Our results support Subthesis S.3.

1.2 Contributions

To support our thesis, we make contributions in theory, methods, implementation and experimentation of MIL. Specifically, our contributions are:

Subthesis S.1

(a) We introduce a novel framework, Active Bayesian MIL, for learning efficient agent strategies with reduced cost of experimentation. Active Bayesian MIL extends Bayesian MIL with an automated experiment selection based on active learning (Section 4.3)

(b) We theoretically evaluate the expected gain in entropy of Active Bayesian MIL compared to Passive Bayesian MIL (Section 4.4)

(c) We provide and describe an implementation of our framework Active Bayesian MIL (Section 4.5)

(d) We experimentally demonstrate over two domains that Active Bayesian MIL converges faster toward efficient agent strategies compared to Passive Bayesian MIL (Section 4.6)

Subthesis S.2

(a) We introduce a new bottom-up method for performing automated predicate invention in MIL (Section 5.3)

(b) We formalise the definition of an equivalence relation for predicates which is used to prune redundant predicates (Section 5.3)

(c) We provide a theoretical proof of the completeness of our bottom-up predicate invention method (Section 5.4)

(d) We derive a theoretical bound over the number of predicate symbols introduced by our predicate invention method (Section 5.4)

(e) We provide and describe an implementation of our method (Section 5.5)

(f) We experimentally demonstrate over two domains that our method can improve learning performance (Section 5.6)

Subthesis S.3

(a) We introduce a new learning system called *MIGO* for learning optimal two-player game strategies for evaluable games (Section 6.3)

(b) We provide and describe an implementation of *MIGO* (Section 6.4)

(c) We experimentally demonstrate over two games that *MIGO* converges faster in terms of Cumulative Minimax Regret than both standard and deep reinforcement learning (Section 6.5)

(d) We experimentally demonstrate that strategies learned by *MIGO* are transferable across different domains (Section 6.5)

(e) We experimentally demonstrate that rules learned by *MIGO* provide some form of comprehensibility (Section 6.5).

In the following, our experimental results will be presented including both the experimental hypotheses tested and the corresponding null hypotheses. We believe both form a crucial part of the experimental design.

1.3 Publications

Parts of the contributions presented in this thesis have been reviewed and published in the following venues:

Parts of Chapter 4 appear in [Hocquette and Muggleton, 2018]. The initial idea of this work, the theoretical framework and the outline of the theoretical analysis are due to the second author Stephen Muggleton. The author of this thesis contributed to (1) formalising the theoretical analysis, (2) conducting the experiments and (3) writing the theoretical analysis, the implementation, the experimental sections and the conclusion.

Parts of Chapter 5 appear in [Hocquette and Muggleton, 2020]. The second author Stephen Muggleton contributed to the initial idea and the theoretical framework. The author of this thesis contributed to (1) the theoretical analysis, in particular proving the completeness of this approach and demonstrating a theoretical bound over the number of predicate symbols introduced, (2) the implementation, (3) conducting the experiments and (4) writing most of the paper.

Parts of Chapter 6 appear in [Muggleton and Hocquette, 2019]. The initial idea of this work and the outline of the theoretical framework are due to the first author Stephen Muggleton. The author of this thesis contributed to (1) formalising the theoretical framework, (2) the implementation, (3) conducting the experiments and (4) writing the theoretical framework, the implementation and experimental sections. Parts of Chapter 6 also appear in [Ai *et al.*, 2021]. The author of this thesis contributed to parts of the experiments and writing parts of the paper.

1.4 Outline

We assume the reader is familiar with general background in Computer Science. For self-consistency, we will however recall some specific concepts relevant to the understanding of our contributions. We have chosen to present our contributions in a structurally simpler order different from the chronological order. At the end, this thesis is organised as follows.

Chapter 2 reviews related work on Computability, Machine Learning, Logic Programming and ILP.

Chapter 3 presents the MIL theoretical framework used throughout this thesis.

Chapter 4 introduces Active Bayesian MIL. Active Bayesian MIL uses active learning for efficient MIL of agent strategies with reduced cost of experimentation. Chapter 4 addresses Subthesis S.1.

Chapter 5 introduces a complete bottom-up method which performs automated predicate invention for improving sample and learning performance in MIL. Chapter 5 addresses Subthesis S.2.

Chapter 6 introduces a new system called *MIGO* for efficient MIL of game strategies for evaluable games. Chapter 6 addresses Subthesis S.3.

Chapter 7 summarises the contributions of this thesis, concludes it and discusses future work.

Each of the Chapters 4, 5 and 6 will be concluded with a future work section addressing possible extensions of the specific contributions presented in each of these Chapters. Conversely, Chapter 7 will include a more general description of global future work related to the claim supported in this thesis.

1.5 Summary

In this Chapter, we have articulated our motivations and objectives. This thesis aims to investigate inductive learning of logic programs from data. We have indicated that this thesis focuses on ILP which is a form of machine learning for inducing hypotheses, as logic programs, from examples. We have clarified our focus in on monotonic learning. We have explained that this thesis more specifically focuses on MIL, a state-of-the-art ILP framework. We have emphasised that MIL approaches have limited sample and learning efficiency. We have claimed that MIL sample and learning efficiency can be improved through 1) methods that revise the instance space (Subthesis S.1), 2) methods that revise the hypothesis space (Subthesis S.2) and 3) methods that revise both the instance space and the hypothesis space (Subthesis S.3). We have highlighted the contributions of this thesis which support our claim, some of which have been published in venues we have stated. We have provided an outline of the organisation of this thesis. The next Chapter reviews works related to this thesis.

Chapter 2

Related Work

This Chapter reviews works relevant to this thesis. Specifically, we focus on related work in Machine Learning, Logic Programming and ILP. We assume the reader has a general Computer Science background.

2.1 Machine Learning

2.1.1 Overview of Machine Learning

Machine learning is a subset of Artificial Intelligence focusing on the study of computer programs which improve their performance automatically over time through experience. Machine Learning is defined as:

Definition 2.1 (Machine Learning System [Mitchell, 1997])**.** *A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E.*

The idea of a learning machine was introduced by Turing [Turing, 1950]. He anticipated the difficulties of achieving human-level AI by manual programming and suggested instead building machines that could learn in the same way as a human child [Muggleton, 2014].

The first Machine Learning program [Samuel, 1959] aimed at learning the game of checkers and succeeded in beating its creator. Samuel referred to "Machine Learning" for describing a system which improves performance through experience rather than being explicitly and manually programmed. Several significant successes of Machine Learning followed in game learning (Atari games [Mnih *et al.*, 2015], Chess [Romstad *et al.*, 2017; Silver *et al.*, 2018], Shogi [Silver *et al.*, 2018], Go [Silver *et al.*, 2016; Silver *et al.*, 2017], Dota 2 [OpenAI *et al.*, 2019], StarCraft [Vinyals *et al.*, 2019]) outperforming the strongest human players.

Machine learning techniques are broadly divided into several categories: supervised, semi-supervised, unsupervised learning and reinforcement learning. In supervised learning, the input training data is labelled. The aim is to learn a function that correctly maps the inputs to their labels. In unsupervised learning, the input training data is unlabelled. The aim is to learn a function that describes the inherent structure of the unlabelled data. In semi-supervised learning, the input training data can be both labelled or unlabelled. The aim is to label unlabelled data using knowledge from labelled data and the structure of all data. In reinforcement learning, the system learns from rewards and penalties instead of labels. Reward and penalties are obtained after executing actions and might be indirect and delayed. The aim is to learn a policy for choosing actions which maximises reward. In the following, we will focus on supervised and semi-supervised learning.

2.1.2 Computational Learning Theory

Computational learning theory studies the feasibility and complexity of learning and proposes computational models describing conditions for successful and efficient learning.

Early work on computational learning theory studied identification in the limit [Gold, 1967]. In this model, the learner is presented with a sequence of training examples, received one by one at each time step. A class of languages is identifiable in the limit if there exists an effective learner which, given any target language of the class and after receiving any sequence of training examples, can correctly identify the target language in a finite amount of time. However, this model of computation only accounts for an exact identification of the target theory in a finite

number of steps and does not accept any approximation. Moreover, this model does not provide any insight into the number of examples and computational resources required for convergence or of the degree of error of output hypotheses.

Probably Approximately Correct (PAC) learning [Valiant, 1984] is a computational complexity based model of learnability. In the PAC learning model, the learner receives examples drawn independently and from the same distribution. The learner aims to find a good approximation of the target theory with high probability and in a polynomially-bounded number of steps:

Definition 2.2 (PAC Learning [Mitchell, 1997])**.** *A concept class C defined over an instance space $\mathcal{X}$ is PAC-learnable by a learner L using an hypothesis space $\mathcal{H}$ if for all $c \in C$, for all distributions D_X over $\mathcal{X}$, for all $\epsilon > 0$ and for all $\delta > 0$, the learner L outputs an hypothesis $h \in \mathcal{H}$ with error at most ϵ and with probability at least $(1 - \delta)$ in time that is polynomial in $\frac{1}{\epsilon}, \frac{1}{\delta}$, the size of instances in $\mathcal{X}$ and the size of c.*

In other words, a concept class is PAC-learnable by a learning system if and only if the learning system outputs with high probability $(1 - \delta)$ an hypothesis that has a small error less than ϵ. This definition implicitly assumes that the learner's hypothesis space $\mathcal{H}$ contains an hypothesis with arbitrarily small error ϵ for every target concept $c \in C$. Moreover, PAC-learning analysis provides lower bounds on the learning performance. These bounds generally are loose compared to empirical performance as they represent worst-case scenario. Finally, another limitation of the PAC-learning model is that it incorporates declarative bias but can not incorporate any assumption about the underlying probability distribution over target concepts.

To overcome these limitations, an average-case model of learning accuracy and sample efficiency for two Bayesian algorithms has been provided [Haussler *et al.*, 1994]. This model depends on properties of both the prior distribution over concepts and the sequence of instances seen by the learner. Similarly, U-learnability (Universal Learnability) [Muggleton and Page, 1994] provides an average-case accuracy, sample and time analysis. U-learnability allows the use of distributional assumptions over hypotheses as parametrised families of possible prior distributions over hypotheses. Conversely to these two models [Haussler *et al.*, 1994; Muggleton and Page, 1994], identification in the limit [Gold, 1967] and the PAC learning

model [Valiant, 1984] only describe conditions for successful learning but these models fail to account for the efficiency of a learning process. The next subsections describe how to evaluate the efficiency of a learning process along two dimensions: the sample efficiency and the learning efficiency. These subsections also present related work aimed at improving the sample efficiency and the learning efficiency.

2.1.3 Sample Efficiency

Sample Complexity A dimension for evaluating the efficiency of a learning system is the data efficiency. Data efficiency is measured by the sample complexity:

Definition 2.3 (Sample Complexity)**.** *Given $\epsilon > 0$, $\delta > 0$, the sample complexity n_{ex} of a learning system L for a concept class C is the minimum number of training samples m needed by L to converge with high probability $1 - \delta$ to a hypothesis with error at most ϵ over concepts drawn from C.*

In the PAC-learning model, the Blumer bound [Blumer *et al.*, 1989] provides an upper bound over the sample complexity as a function of the size of the hypothesis space. Assuming $\epsilon > 0$, $\delta > 0$, the Blumer bound states that the number of training examples n_{ex} required to PAC-learn an hypothesis with error at most ϵ and with probability at least $1 - \delta$ from an hypothesis space of size $\mid \mathcal{H} \mid$ is:

$$n_{ex} \geq \frac{1}{\epsilon}(ln(\mid \mathcal{H} \mid) + ln(\frac{1}{\delta}) + ln(c)) \text{ with c constant} \tag{2.1}$$

The Blumer bound can be interpreted as the fact that any hypothesis which explains a sufficiently large number of training data will generalise well over unobserved examples with high probability. The Blumer bound falls within the PAC-learning model and therefore assumes that a good approximation of the target hypothesis is within the learner's hypothesis space. The Blumer bound indicates that, when several hypotheses spaces verify this assumption, searching the smaller space will result in higher predictive accuracies compared to searching the larger space. In that sense, the availability of more prior knowledge can reduce the hypothesis space thus improve the generalisation performance of the learner.

This model of computation assumes examples are sampled independently at random.The sample complexity also can be improved by considering different learning protocols, such as active learning or reinforcement learning.

Active Learning Active learning is motivated by learning problems in which unlabelled data is abundant but labels are expensive to obtain. An active learner seeks for low label complexity: it aims to learn a model with high accuracy while minimizing the cost of labelling data. To do so, an active learner can choose unlabelled data, also called instances, and query their label to an oracle. Once labelled, an instance is called an example. The objective in active learning is to learn a model with high performance while making fewer queries than the number of random labels required by a passive learner to achieve the same accuracy. Active learning has been widely adopted in various machine learning tasks including natural language processing [Thompson *et al.*, 1999], biological experimentation [King *et al.*, 2004], computer vision [Vijayanarasimhan and Grauman, 2009]. Different query scenarios have been introduced [Settles, 2009; Hino, 2020].

In the membership scenario, the learner has access to an oracle which can provide on request the label of any points from the instance space, even artificially generated ones [Angluin, 1988]. However, freely generated instances may be unnatural and uninterpretable to humans oracle thus hindering their labelling. To alleviate this issue, newly generated instances can be restricted to local queries, which are queries of instances close to training instances [Awasthi *et al.*, 2013]. A variant is learning with equivalence queries [Angluin, 1988]: instead of generating instances, the learner generates and presents an hypothesis from the target concept class to the oracle. The oracle either validates or invalidates this hypothesis. In the latter case, the oracle also provides a counter-example. In stream-based selective sampling, the learner samples data from the instance distribution and decides whether to label or discard each sampled instance [Cohn *et al.*, 1994; Freund *et al.*, 1997]. Several criteria may be used for deciding whether or not to query the label of an instance. The learner may for instance evaluate samples according to some informativeness measure and query samples which evaluation is above some threshold or use these evaluations to make biased random decisions. In pool-based active learning,

the learner has access to a large set of initial unlabelled instances. The learner successively chooses instances among this set and queries their label to an oracle [Lewis and Gale, 1994]. Instances typically are chosen greedily according to some informativeness measure evaluated over all instances in the pool.

These query scenarios involve strategies for evaluating the informativeness of instances. The learner for instance can compute an explicit region of uncertainty, which is the subpart of the instance space on which competing hypotheses disagree. It only queries sampled instances that fall within this region in stream-based selective sampling [Cohn *et al.*, 1994]. Alternatively, informativeness can be evaluated and scored using measures such as entropy [Shannon, 1948], uncertainty sampling [Lewis and Gale, 1994], least confident sampling [Culotta and McCallum, 2005] and margin sampling [Scheffer *et al.*, 2001].

We say that an hypothesis is consistent with some examples if it correctly predicts their labels. An active learner aims to obtain information through labels to discriminate between competing hypotheses consistent with the examples seen so far. This set of competing consistent hypotheses is called the version space and is defined as follows:

Definition 2.4. *Assume an hypothesis space $\mathcal{H}$ and a set of training examples E, the version space V is the subset of hypotheses from $\mathcal{H}$ consistent with the training examples E.*

The version space contains all versions of the target concept that are plausible so far given the training set. One can evaluate the shrinkage of the version space during the learning process to measure the benefits of active learning over passive learning. Several measures have been suggested for evaluating the size of the version space as a function of number of labelled queried. These measures include the diameter of the version space [Dasgupta, 2005b; Tosh and Dasgupta, 2017], the measure of the size of the region of disagreement [Hanneke, 2007; Hanneke, 2014], the metric entropy [Kulkarni *et al.*, 1993] and the size of the version space [Mitchell, 1982; Dasgupta, 2005a]. For instance, in the version space learning algorithm, the learns maintains a representation of the version space. The instances whose labels should be requested are the instances which come closest to being consistent with exactly half of the hypotheses in the version space [Mitchell, 1982]. These instances provide the highest expected

information gain: regardless of their classification, finding out their label will allow rejecting the closest to one-half of the current competing hypotheses. Therefore, in the optimal case and for a finite hypothesis space $\mathcal{H}$ of size $| \mathcal{H} |$, the correct target hypothesis can be found with only $log(| \mathcal{H} |)$ experiments when following this query strategy and assuming the target hypothesis is contained in the hypothesis space $\mathcal{H}$. In Chapter 4, we will investigate an active learner based upon this idea. We consider an extension to cope with potentially infinite hypothesis spaces and aimed at learning agent strategies.

Reinforcement Learning Reinforcement learning [Sutton and Barto, 2018] considers an autonomous agent interacting with an environment. The agent is equipped with sensors to observe the state of its environment and it is capable of performing actions to alter the state of its environment. The agent might receive rewards or penalties after performing some actions. Rewards and penalties indicate the desirability of the resulting states. Goals are specified as high-reward states. The reinforcement learning problem is for the agent to learn a policy for choosing actions such as to maximise the total reward received and thus to achieve goals [Mitchell, 1997]. Conversely to active learning, the agent receives rewards which may be indirect and delayed instead of instance labels. Moreover, the task is to learn a policy maximising the cumulative reward rather than to learn a label function of the inputs. As in active learning, the agent influences the distribution of training examples received. A reinforcement agent obtains training examples depending on the sequence of actions it chooses. Therefore, the choice of experimentation strategy is crucial for effective learning. The agent should prefer actions that it has already tried and that are known to yield good rewards. However, the agent first must discover such actions. Besides, committing too early to good actions might imply missing out on even better actions. In this sense, the learner faces a trade-off between exploitation and exploration. It must balance the exploitation of states and actions that are known to yield high-reward and the exploration of unknown states and actions for obtaining new information. Another specificity of reinforcement learning is that rewards feedbacks are indirect and delayed: rewards may only be credited after a sequence of moves. Therefore, it is unclear the degree to which each move in the sequence deserves credit or blame for the rewards received. In this

sense, the agent faces the temporal credit assignment problem [Minsky, 1961], which is the problem of how to assign credit for the success among the many decisions that were involved.

Reinforcement learning has been widely studied in the context of game-playing. MENACE (Matchbox Educable Noughts-And-Crosses Engine) [Michie, 1963] was the world's earliest reinforcement learning system and was specifically designed to learn to play Noughts-and-Crosses. An early manual version of MENACE used a stack of matchboxes. Each box represented an accessible board position and contained coloured beads representing possible moves. Moves were selected by randomly drawing a bead from the box representing the current board. After having completed a game, MENACE's punishment or reward consisted of changing the number of beads for the colours drawn in the boxes used during the game. Beads were added or subtracted depending on the outcome of the game. This modified the probability of the selected moves being played again in these board positions [Brooks, 2017]. HER (Hexapawn Educational Robot) [Gardner, 1962] is a similar system for the game of Hexapawn. Samuel's program [Samuel, 1959] learns the game of checkers through self-play. It plays by performing a look-ahead search based on Shannon's minimax procedure [Shannon, 1950]. Each position is given the score of the position that would result from the best move, assuming that the opponent always chooses the move that minimises the learner's score while the learner always chooses the move that maximises its score. Samuel's learning program also uses heuristics to determine how to expand the search tree and when to stop searching. Time efficiency was gained through rote learning: the program saves all board positions encountered together with their computed scores.

More generally, Q-learning [Watkins, 1989] addresses the problem of learning an optimal policy from delayed rewards and by trial and error. The learned policy takes the form of Q-values associated with each action available from each state. These Q-values represent the maximum expected discounted reward achievable if executing the action considered from the state considered. Q-learning is an algorithm that approximates the Q-values by successively updating them according to the rewards received and the maximum discounted expected future rewards. Q-learning does not require any prior knowledge about the environment and considers the reward and action transition functions are unknown. A guarantee of asymptotic convergence to

optimal behaviour has been proved for systems with bounded rewards and when each state-action pair can be visited infinitely often [Watkins and Dayan, 1992]. Under these assumptions, the convergence is guaranteed regardless of the training sequences chosen. However, choosing suitable training sequences might improve the speed of convergence thus the learning efficiency. A limitation of Q-learning is that the number of parameters to learn is the number of state-action pairs. Q-learning presents little generalisation ability across pairs which highly limits the sample efficiency as we will see in Chapter 6.

Deep reinforcement learning is an extension of Q-learning that uses a deep convolutional neural network to approximate the Q-values. A deep Q-learner takes as input states and outputs the predicted Q-value for all possible actions. Deep reinforcement learning systems provide better generalisation ability and scalability which has been demonstrated through a diverse range of tasks from the Atari 2600 games [Mnih *et al.*, 2015], the game of Go [Silver *et al.*, 2016; Silver *et al.*, 2017], Chess and Shogi [Silver *et al.*, 2018], Dota 2 [OpenAI *et al.*, 2019], StarCraft [Vinyals *et al.*, 2019]. However, these systems have inherent drawbacks. First, they suffer from data inefficiency and generally require the execution of many games to converge. Moreover, these systems lack the ability to transfer the knowledge learned to unseen domains. A change in the inputs or goals, although minor, such as changing the colour or the size of an object in a game, requires significant retraining. By contrast, a strength of human intelligence is the ability to learn models and use them for radically different tasks [Lake *et al.*, 2017]. Finally, the learned strategy is implicitly encoded into the Q-value parameters which makes their operation largely opaque to humans and little interpretable [Marcus, 2018].

Recent works have intended to explain the policies learned [Zahavy *et al.*, 2016]. Deep symbolic reinforcement learning [Garnelo *et al.*, 2016] combines neural network learning with aspects of symbolic AI to overcome the aforementioned shortcomings: a neural back-end builds a symbolic representation, then a symbolic front end learns a policy in the form of Q-values. Relational reinforcement learning [Džeroski *et al.*, 2001; Tadepalli *et al.*, 2004; Kersting *et al.*, 2004] is a reinforcement learning framework where states, actions and policies are represented relationally. Relational reinforcement learning systems benefit from background knowledge and declarative bias. They learn a Q-function using a relational regression algorithm [Driessens *et al.*, 2001;

Driessens and Ramon, 2003]. State, actions and learned policies have a relational representation which allows to abstract over specific object identities as well as the number of objects involved and simplifies transfer between tasks [Croonenborghs *et al.*, 2008]. Hierarchical reinforcement learning decomposes a reinforcement learning problem into a hierarchical process such that higher levels correspond to high-level goals and lower levels to their execution. RePReL [Kokel *et al.*, 2021] combines a high-level hierarchical relational learner with a low-level reinforcement learner. The former plans a sequence of subgoals to achieve its goal and the latter learns a policy to achieve each of these identified sub-goals. Similarly, the ILP system δILP has been paired with Q-learning, the former learns logical rules describing the state transitions between subtasks and the latter learns a policy for each subtask [Xu and Fekri, 2021]. These systems achieve state abstraction which provides improved data efficiency, better generalization to unseen tasks and better interpretability.

By contrast, we will focus in Chapter 6 on devising a purely symbolic system addressing these challenges.

2.1.4 Learning Efficiency

Time Complexity Besides sample efficiency, another dimension for evaluating the efficiency of a learning system is time efficiency. Time efficiency is measured by the time complexity:

Definition 2.5 (Time Complexity)**.** *Given $\epsilon > 0$, $\delta > 0$, the time complexity T of a learning system L for a concept class C is the minimum number of elementary computer operations needed by L to converge with high probability $1 - \delta$ to a hypothesis with error at most ϵ over concepts drawn from C.*

The time complexity is estimated by counting the number of elementary operations performed by an algorithm, supposing that each elementary operation takes an equal fixed amount of time to perform. For instance, the time complexity of a deterministic Turing machine is the total number of operations the machine executes before it halts and outputs an answer. Time

complexity usually is expressed as a function of the size of the input and one commonly focuses on the asymptotic behaviour of the complexity when the input size increases.

Generalisation as Search The process of generalisation can be characterised as a search problem [Mitchell, 1982]: the task is to identify hypotheses within a search space that are consistent with the training instances. The search space, or hypothesis space, is defined by the generalisation language. Therefore, the complexity of the search is a function of the complexity of the representation. Methods for generalisation are characterised by the search strategies they employ. Generate-and-test strategies generate new hypotheses independently of the input data. These candidate hypotheses are then tested against the training set. By opposition, data-driven strategies successively revise current hypotheses to eliminate inconsistencies between these current hypotheses and the training data.

For any of these strategies, traversing this entire hypothesis space by explicitly enumerating every hypothesis not only is inefficient for large hypothesis spaces but also can be infeasible for infinite hypothesis spaces. However, the search can more efficiently be organised by taking advantage of a general-to-specific ordering of hypotheses [Mitchell, 1997]. Informally, given two hypotheses H_1 and H_2, H_1 is more general than or equal to H_2 if and only if any instance that satisfies H_2 also satisfies H_1. In this case, H_2 is said to be more specific than H_1. For instance, the bottom clause [Muggleton, 1995] is the most specific clause that entails an example. The bottom clause can be used to constrain thus facilitate the search.

The version space strategy is a data-driven strategy. It involves maintaining a representation of the version space, which is the set of all hypotheses consistent with the observed training instances and describable within the given generalisation language. Version space algorithms can cope with infinite hypothesis spaces when they use a compact description of the version space which does not require explicitly enumerating all of its members. For instance, the version space can be represented by two sets representing its most general and least general members respectively [Mitchell, 1977]. In Chapter 4, we will represent the version space with sampled representative members [Muggleton and Tamaddoni-Nezhad, 2008; Muggleton *et al.*, 2013].

Inductive Bias Inductive bias refers to the set of assumptions made by a learning algorithm in order to induce an hypothesis. These assumptions might be explicitly stated or implicitly encoded. These assumptions represent any basis for choosing one hypothesis over another, other than consistency with the training examples [Mitchell, 1980]. The inductive bias defines the hypothesis space. Introducing more bias reduces the size of the hypothesis space thus improves learning performance. According to the Blumer bound described in Section 2.1.3, a stronger bias in turn can provide lower predictive errors, assuming the target hypothesis is not excluded from the hypothesis space. However, when a bias is too strong, it might be restrictive to the point it rules out the correct hypotheses and their approximations from the hypothesis space. In this sense, there is a trade-off between the search complexity, which is improved by a small search space thus a strong bias, and the availability of a correct theory, which is improved by a large search space thus a weak bias. In the extreme case, a learner with no bias makes no a priori assumptions about the target hypothesis. Its unbiased hypothesis space is infinite and contains all hypotheses. All consistent hypotheses in this unbiased hypothesis space are treated equally with no preference. Therefore, there is no rational basis for generalisation leap. In that sense, bias-free learning is futile [Mitchell, 1997].

Several kind of bias are usually used. First, the use of prior knowledge about the domain, or background knowledge, can provide strong and justifiable constraints on the hypothesis space [Mitchell, 1980]. Second, the search bias specifies the way the system searches through the hypothesis space: it implements preference but no hard restrictions on hypotheses. Finally, the language bias specifies the hypotheses that are allowed in the search space. It imposes hard restrictions over the range of hypotheses that are expressible and therefore that can be learned. For instance, a common bias is to bound the complexity of learned hypotheses. Occam's razor is a principle which formalises this bias. It states that the simpler hypothesis that fits the data should be preferred. This razor can be justified by the fact that simpler models, in general, are easier to understand, remember and use for humans, and easier to manipulate and store for computers. However, the size of an hypothesis is representation-dependent, which implies that learners with different representations can prefer different hypotheses. Moreover, greater simplicity does not necessarily lead to greater accuracy [Domingos, 1999].

Learning algorithms have traditionally relied on static and hand-designed bias. A challenge is to automatically identify appropriate biases. These biases can be learned from previous learning experience, which is a way to perform automated bias shift [Sutton, 1992]. In Chapters 5 and 6, we will investigate methods to acquire appropriate bias to improve learning efficiency.

2.2 Logic Programming

Logic Programming is a subset of Computational Logic. Logic programming concerns the use of logic for representing and solving problems [Kowalski, 1979]. Logic Programming is a declarative programming paradigm and is based on formal logic. Programs are logical theories, computations and reasoning are achieved with deductive logical inferences over these theories. For instance, resolution [Robinson, 1965] is a method for automated deduction using a single rule of inference. Resolution is sound and complete for deductive inference in first-order logic. Resolution uses unification to operate directly on first-order logic sentences. SLD-resolution [Kowalski and Kuehner, 1971] is a more efficient inference method restricted to Horn logic. Horn logic is a subset of first-order logic which still is Turing-complete [Tärnlund, 1977]. SLD-resolution also is sound and complete for Horn logic.

Most common logic programming languages are Prolog, Datalog and Answer Set Programming (ASP). Prolog [Colmerauer and Roussel, 1996; Sterling and Shapiro, 1994; Bratko, 2012] is based on Horn logic and is a Turing-complete language. Computations within Prolog are the search for logical proofs based on the application of SLD-resolution. Prolog uses a sequential last-in-first-out backtracking execution strategy. Given a goal, multiple unifications during resolution might be possible which creates choice points. At each choice point, these alternative unifications leading to different sub-goals are successively considered, one at a time, via backtracking. Prolog is not purely declarative, the execution depends on the order of clauses due to the use of extra-logical features such as cuts. Prolog includes negation as failure which allows for non-monotonic reasoning. Datalog [Ceri *et al.*, 1989] syntactically is a subset of Prolog submitted to several restrictions. First, every variable that appears in the head of a clause must

also appear in a non-negated body literal of the same clause. Also, complex terms, such as terms involving function symbols, are disallowed. Finally, the use of negation and recursion are subject to stratification restrictions. Compared to Prolog, Datalog trades Turing-completeness for decidability. Moreover, Datalog does not allow Prolog's cut operator and is fully declarative. Finally, ASP is based on stable model semantics [Gelfond and Lifschitz, 1988]. While a definite logic program has only one model, an ASP program can have no model or multiple models called answer sets. Compared to Datalog and Prolog, ASP is more expressive, and allows for instance aggregates, disjunctions in the head of clauses, choice rules, hard and weak constraints.

2.3 Inductive Logic Programming

2.3.1 Logical Reasoning

Reasoning is the process of using existing knowledge to derive new knowledge. In logical reasoning, the initial knowledge is called the premises and new knowledge derived are called the conclusions. Premises and conclusions are logical sentences. The three major methods of reasoning are deductive, abductive and inductive reasoning.

Deduction is the process of deriving the consequences of general statements to reach specific conclusions. If the premises are true, the conclusions necessarily are true. Abduction is the process of explaining observations, that is of reasoning from effects to possible causes. Induction is the process of deriving reliable generalisations from observations. Abduction usually is regarded as the process of providing an explanation in the form of a set of ground facts while induction is the process of providing an explanation in the form of a set of universally quantified rules [Muggleton *et al.*, 2014]. Unlike deductive reasoning, abductive and inductive reasoning derive conclusions which are not guaranteed to be correct. Whereas deduction aims to make explicit some conclusions already implicit the premises, abduction and induction aim to discover new knowledge from observations. Both abduction and induction can be viewed as the inverse of deduction. In the following, we investigate techniques that combine these three

major methods of logical reasoning.

2.3.2 ILP

ILP [Muggleton, 1991; Nienhuys-Cheng, 1997; Cropper and Dumančić, 2022] is a form of machine learning which uses logic programming to represent knowledge. An ILP learner takes as input some background knowledge and a set of training examples containing positive and negative examples. The goal of an ILP learner is to induce an hypothesis as a logic program which, together with the background knowledge, is consistent with the training examples and generalises beyond them.

ILP benefits from several strengths. First, a distinctive feature of ILP systems is their ability to make use of background knowledge. The importance of using initial built-in knowledge, or background knowledge and the difficulty of learning without it has been pointed early on [Turing, 1950] and is essential for achieving robust intelligence [Marcus and Davis, 2019]. ILP systems can easily make use of prior domain knowledge and benefit from a strong explicit inductive bias which provides a high generalisation capability. ILP systems are data efficient and typically can generalise well from few examples only, often a single example [Lin *et al.*, 2014; Dai *et al.*, 2017]. Moreover, ILP systems represent the examples, the background knowledge and learned hypotheses as logic programs. The logical representation provides high expressivity to ILP systems which allows learning complex relational theories involving recursions and high-level reasoning [Cropper *et al.*, 2020a]. Also, since learned hypotheses are represented in the same language as the background knowledge, they can easily be explicitly stored, remembered and then reused. Therefore, ILP systems naturally support lifelong learning and transfer learning [Lin *et al.*, 2014]. Finally, another strength of ILP systems is their ability to generate human-interpretable explanations. It is usually straightforward to translate logic programs into a series of understandable sentences [Muggleton *et al.*, 2012]. Learned models thus can be read and examined by humans, which is crucial for explainable AI. Conversely, poor generalisation ability, the necessity for large training sets and limited comprehensibility precisely are common limitations of other machine learning techniques [Chollet, 2019;

	Inductive Bias	Predicate Invention	Learning of recursion	Hypothesis search
Progol	mode declaration	✗	partly	bottom-up / top-down
TILDE	mode declaration	partly	✗	top-down
Aleph	mode declaration	✗	partly	bottom-up / top-down
ILASP	mode declaration	partly	✓	meta-level
Metagol	meta-rules	✓	✓	top-down
δILP	program templates	partly	✓	meta-level
HEXMIL	meta-rules	✓	✓	meta-level
Popper	declarations	✓	✓	meta-level

Table 2.1: Comparison of state-of-the-art ILP systems

Marcus and Davis, 2019] when learning rich representations from few examples is characteristic of human intelligence [Lake *et al.*, 2017].

These strengths have allowed ILP systems to achieve results in a wide range of challenging applications including learning game strategy [Bain and Muggleton, 1994], robot scientist [King *et al.*, 2004], modeling inhibition in metabolic networks [Tamaddoni-Nezhad *et al.*, 2006], automated discovery of food web [Bohan *et al.*, 2011], learning regular grammar [Muggleton *et al.*, 2014], string transformation programs [Lin *et al.*, 2014], robot strategies [Cropper and Muggleton, 2015], automated reconstruction of ecological networks [Bohan *et al.*, 2017], logical vision [Dai *et al.*, 2017], Petri nets models of biological systems [Bain and Srinivasan, 2018] and code search [Sivaraman *et al.*, 2019].

We discuss in the following subsections features of ILP that support the strengths aforementioned. We compare state-of-the-art ILP systems along some of these features relevant to this thesis. An overview of this comparison is presented in Table 2.1. We specifically focus on comparing Metagol, on which this thesis is based, against related state-of-the art systems.

2.3.3 Inductive Bias

ILP systems make use of inductive bias to constrain and restrict the search. As explained in Section 2.1.4, the inductive bias guides the search for consistent hypotheses and helps making the search tractable. Language bias is a common ILP inductive bias: it specifies the hypotheses allowed in the search space. Language bias in ILP is also called declarative bias and is explicitly expressed logically. The language bias can be characterised as semantic bias, which

imposes restrictions on the behaviour of induced hypotheses and syntactic bias, which imposes restrictions on the form of clauses allowed in hypotheses [Adé *et al.*, 1995]. Different representations have been suggested to encode the syntactic declarative bias in ILP. For instance, syntactic declarative bias has been specified by giving a grammar of the permitted clauses [Cohen, 1995]. Also, mode declarations specify which predicate symbols may appear in the head or body of clauses, together with optional bounds over their number of occurrence in a clause, the types of their arguments, and whether their arguments must be ground [Muggleton, 1995; Blockeel and De Raedt, 1998; Srinivasan, 2001; Corapi *et al.*, 2011; Law, 2018]. Alternatively, meta-rules is another common declarative bias. Meta-rules are higher-order clauses which act as program templates. A meta-rule specifies the form of clauses in the hypothesis space including the number of body literals, the respective arity of head and body literals, the number and binding of first-order variables. Early versions of meta-rules, also called second-order schema, have been introduced early on [Emde *et al.*, 1983; Kietz and Wrobel, 1992; De Raedt and Bruynooghe, 1992; Flener, 1996]. As shown in Figure 2.1, *Metagol* [Muggleton *et al.*, 2015; Cropper and Muggleton, 2016] and HEXMIL [Kaminski *et al.*, 2018] both use meta-rules. Compared to mode declarations, meta-rules impose a stronger bias. Conversely to mode declarations or grammars, meta-rules used in MIL are logical statements, which provides the potential for reasoning about them and manipulating them alongside first-order background knowledge [Muggleton *et al.*, 2015]. We will use meta-rules as language bias in this thesis. δ-ILP uses rule templates which similarly describe the set of rules which can be generated. Finally, Popper [Cropper and Morel, 2021] uses declarations, which specify the predicate symbols that may appear in the head or body of clauses, the maximum number of variables and literals in clauses and the maximum number of clauses.

Choosing an appropriate language bias is crucial for realisable and tractable learning. The language bias must be expressive enough to allow the representation of consistent hypotheses in the hypothesis space. However, the language bias must be precise enough to restrain the search space and make the search tractable. The choice of appropriate language bias has traditionally relied on user choices based on knowledge about the domain. Recent work have focused on theoretically identifying general bias as irreducible sets of meta-rules sufficiently expressive to

allow induction of all programs in some fragments of dyadic logic [Cropper and Muggleton, 2014; Cropper and Tourret, 2020]. Alternatively, language bias can be learned. For instance, meta-knowledge about type, mode or predicate symmetry can be extracted from the data [McCreath and Sharma, 1995]. Alternatively, constraints over the hypothesis space can be learned across multiple tasks [Bridewell and Todorovski, 2007]. Finally, language bias shift can automatically be performed when the language bias is not sufficient to describe a consistent hypothesis. For instance, Clint changes its bias from a series of built-in description languages when necessary [De Raedt and Bruynooghe, 1992].

2.3.4 Background Knowledge

ILP systems have the distinctive ability to make use of background knowledge. The background knowledge is provided as input in the form of a logic program. It is a set of relational predicates explicitly defined and which provide information about the domain studied. The background knowledge is a form of inductive bias which restricts the hypothesis space, guides the search for hypotheses thus facilitates learning [Gulwani *et al.*, 2015].

Given a problem, specific background knowledge can be hand-written by domain experts, which tailor and carefully choose which knowledge to include. However, manual selection of appropriate background knowledge can be time-consuming, expensive and cumbersome for users. Moreover, the reliance on hand-written background knowledge prevents straightforward extrapolation of systems across different domains. Finally, the background knowledge must be carefully selected and its choice is crucial for learning performance. Indeed, the size of the hypothesis space is a function of the size of the background knowledge. A large background knowledge with irrelevant information can hinder the search for good hypotheses [Srinivasan *et al.*, 2003]. Conversely, too few background knowledge may lead to an expensive construction of a consistent hypothesis or may not even allow for the representation of a consistent hypothesis. In that sense, selection of adequate and relevant background knowledge is determinant and can improve learning performance. A challenge is to find a suitable trade-off between a large background knowledge which allows to represent a multitude of hypotheses and to solve a

multitude of problems, but limited enough such that the learner is not overwhelmed [Cropper, 2020].

For instance, selection of relevant background knowledge has been evaluated based on generality of background predicates [Patsantzis and Muggleton, 2018], statistical or syntactical criteria [Cropper, 2020]. Deepcoder [Balog *et al.*, 2017] trains a neural network to predict the probability of background predicates appearing in the program that generated the examples. These predictions are used to guide the search for consistent hypotheses. DreamCoder [Ellis *et al.*, 2018] learns via bootstrapping new and reusable knowledge that is shared in a multi-task setting while jointly training a neural network to efficiently predict the posterior of programs in the resulting search space.

In Chapter 5, we extend the learner's language with predicate invention: the background knowledge is augmented with an extension of the immediate consequence operator. Selection of relevant predicates is performed in a logical setting, and is based upon equivalence of logic programs from success sets.

2.3.5 Predicate Invention

Predicate invention is the automatic introduction of new predicates in the language. Instead of relying exclusively on predicates provided as part of the background knowledge, a learner which can perform predicate invention can invent new definitions thus augmenting its available vocabulary. Predicate invention is a way of performing automated discovery of new reusable sub-concepts [Muggleton *et al.*, 2012]. These sub-concepts can be combined to build hierarchies of structured components that represent increasingly abstract and complex ideas. This ability to construct complex concepts through hierarchical combinations of primitive features is a key characteristic of human intelligence [Kurzweil, 2013] and is crucial for developing intelligent systems [Kramer, 2020].

Predicate invention has two main uses when learning [Kramer, 1995]. First, the introduction of new predicates can be used for reformulating a program, thus providing a simpler and more

compact representation within the target language [Flach, 1993; Stahl, 1993]. Simpler representations in general are more structured and more human-readable. Also, simpler representations are easier to learn according to the Blumer bound [Blumer *et al.*, 1989] since a smaller space needs to be searched. In this sense, predicate invention, by providing access to simpler representations, can help improving learning performance [Cropper, 2019]. Second, predicate invention has been investigated as bias shift [Stahl, 1995] for overcoming the limitations of insufficient vocabulary for learning. The introduction of new predicates extends the hypothesis language which in turn broadens the range of expressible and thus learnable concepts. In this sense, predicate invention may allow to learn hypotheses for tasks which are unsolvable in the initial language. Predicate invention is a hard challenge in ILP due to its high combinatorial complexity [Kramer, 1995; Muggleton *et al.*, 2012]. The construction of irrelevant predicates may be useless, or even affect learning performance, therefore one must detect situations in which predicate invention can be useful and should be performed [Stahl, 1994]. Also, the number of predicates that can potentially be constructed is very large since invented predicates may have varying arities, arguments and definitions. Therefore only a small selected subset of invented predicates can practically be constructed. Finally, the predicates generated may be evaluated and non-useful ones discarded [Kramer, 1995].

As shown in Table 2.1, early ILP systems, such as Progol [Muggleton, 1995] or Aleph [Srinivasan, 2001] do not support predicate invention. CIGOL [Muggleton and Buntine, 1988] uses an early predicate invention approach based on the use of W operators within the inverting resolution framework. CIGOL performs interactive predicate validation: the system asks the user to validate and name new invented predicates which limits their number and guarantees some level of relevance and comprehensibility. Compression is a major criterion for generating invented predicates [Kramer, 1995]. Early ILP approaches identified similar patterns in the data that are not due to chance and exploited compression of hypotheses to form more compact theories through the generation of new concepts. For instance, Duce [Muggleton, 1987] and CIGOL [Muggleton and Buntine, 1988] prefer operators on the basis of their ability to shorten the minimal achievable encoding of the examples. FRINGE [Pagallo and Haussler, 1990] constructs boolean features by searching for repeated occurrences at the fringe of deci-

sion trees. CHAMP [Kijsirikul *et al.*, 1992] constructs a new predicate when no clause can be learned from the available predicates. In this case, it identifies a minimal variable set and uses it to form a new predicate that discriminates between positive and negative examples while reducing the encoding of positive examples. Statistical predicate invention [Kok and Domingos, 2007] automatically invents predicates which compress the data by means of multiple relational clusterings of the object, attribute and relation symbols in the data. It iteratively refines these clusters of symbols based on the clusters of symbols they appear in atoms with. CUR^{2}LED [Dumančić and Blockeel, 2017] forms predicates from clustering constants and relations in the background knowledge according to various similarity measures. Alps [Dumančić *et al.*, 2019] performs predicate invention in a way inspired by neural auto-encoders. An encoding logic program mapping the input data to a compressed latent symbolic representation is learned. A decoding logic program reconstructs the data from its latent representation. Knorf [Dumancic *et al.*, 2021] compresses learnt programs by removing redundancies in them. Knorf in particular revises invented predicates and introduces new ones, such as to optimise the representation of knowledge.

A restricted form of predicate invention, called prescriptive predicate invention, relies on user-provided specification of the arity and argument types of the new predicates [Corapi *et al.*, 2011; Law, 2018; Evans and Grefenstette, 2018]. Conversely, automated predicate invention does not require the user to define the structure of invented predicates. For instance, MIL systems [Muggleton *et al.*, 2014; Muggleton *et al.*, 2015; Kaminski *et al.*, 2018] achieve automated predicate invention from the use of meta-rules by introducing new predicate symbols during the construction of meta-substitutions. In Chapter 5, we extend MIL with an automated predicate invention method. Predicates are invented bottom-up from the background knowledge. We demonstrate it can improve learning performance. Predicate invention in MIL is also related to predicate invention realised by meta-level abduction [Inoue *et al.*, 2009]. Compared to meta-level abduction, MIL introduces new predicate symbols which represent relations rather than new objects or propositions.

Finally, an algorithm related to predicate invention is reformation [Bundy and Mitrovic, 2016]. Reformation automatically changes the language to repair faulty logical theories. Typical lan-

guage changes include splitting or merging of predicates and changing the arity of predicates.

2.3.6 Learning of Recursion

A recursive logic program is a program in which the same predicate appears both in a head and a body literal in a same rule. Recursions allow for compact representations with high generalisation ability: recursive hypotheses have finite size but can generalise to arbitrary input size and are applicable to an infinite set of atoms [Cropper and Dumančić, 2022]. In this sense, recursion provides high expressivity. Systems that support recursion can generalise from small numbers of examples, often a single one [Lin *et al.*, 2014; Cropper, 2019]. Conversely, systems which can not learn recursions need to learn a separate definition for each possible depth of execution. The ability to learn recursions thus is valuable for a wide range of applications [Muggleton *et al.*, 2014; Lin *et al.*, 2014; Cropper, 2019] but is considered as a difficult problem in ILP [Muggleton *et al.*, 2012; Cropper *et al.*, 2020a].

Early ILP systems, such as GOLEM [Muggleton and Feng, 1990] or TILDE [Blockeel and De Raedt, 1998] cannot support recursion. As shown in Table 2.1, some systems, such as Progol [Muggleton, 1995] or Aleph [Srinivasan, 2001], only support limited recursion. They require examples of the base and inductive cases in that order. Also, they can not learn mutually recursive definitions of different relations. By opposition, recent systems based on MIL (*Metagol* [Muggleton *et al.*, 2015], HEXMIL [Kaminski *et al.*, 2018]) rely on the use of meta-rules and can learn recursive programs thanks to the use of recursive meta-rules. Other recent ILP systems (δILP [Evans and Grefenstette, 2018], ILASP systems [Law, 2018], Popper [Cropper and Morel, 2021]) also support efficient learning of recursive logic programs. In the following Chapters, we learn hypotheses such as agent strategies or string transformations involving recursion.

2.3.7 Hypothesis Search

ILP systems search for a consistent hypothesis in the hypothesis space defined by the language bias. As explained in Section 2.1.4, the search can be made more efficient by making use of the ordering of hypotheses imposed by a generality relation. Having ordered the hypothesis space, usual search approaches are bottom-up and top-down. Bottom-up algorithms start from specific hypotheses from the examples and then generalise them. For instance, the propositional learner Duce [Muggleton, 1987] begins with a set of specific rules, one for each positive example. This set is successively generalised using six operators: at each iteration the generalisation which produces greater compression of the rule set is chosen. CIGOL [Muggleton and Buntine, 1988] generalises first-order rules using inverse resolution. Golem [Muggleton and Feng, 1990]) successively generalises pairs of examples using relative least-general generalisation. XHAIL [Ray, 2009] performs abductive, deductive, and inductive inference for generalising an initial ground specific hypothesis called Kernel Set. Conversely, top-down algorithms first compute a most general hypothesis and then specialise it. FOIL [Quinlan, 1990] successively forms clauses by successively adding body literals according to an information heuristic until no negative examples are covered. TILDE [Blockeel and De Raedt, 1998] starts with the most general program, the empty one, and successively specialises it by adding decision nodes with the highest information gain thus splitting the training set. Finally, variants of bidirectional search were implemented in algorithms alternating generalisation and specialisation steps to search through the lattice of clauses [Fensel and Wiese, 1993; Zelle *et al.*, 1994; Srinivasan, 2001; Califf and Mooney, 2003]. For instance, Progol [Muggleton, 1995] constructs a most specific clause, the bottom clause, that entails an uncovered positive example. Next, Progol performs a top-down search by employing a variant of the A* search to find the best possible consistent definition in the search space bounded by the bottom clause. Aleph employs a similar combination of bottom-up and top-down strategies. Similarly, Fastlas [Law *et al.*, 2020] first identifies a subset of the hypothesis space from specific rules covering single examples. It then uses a meta-level approach to search this space.

Finally, recent ILP learners employ a meta-level search. For example, the search can be for-

mulated as an ASP problem and be delegated to an ASP solver (ASPAL [Corapi *et al.*, 2011], ILASP [Law, 2018], HEXMIL [Kaminski *et al.*, 2018], Popper [Cropper and Morel, 2021], the Apperception Engine [Evans *et al.*, 2021]). δ-ILP [Evans and Grefenstette, 2018] implements the ILP problem as a differentiable neural-based architecture.

2.3.8 Lifelong Learning

Lifelong learning [Mitchell *et al.*, 2018] is a paradigm is which the learner is presented with a collection of tasks and learns cumulatively: knowledge acquired from solving one problem can be used to help learning more effectively subsequent tasks. Lifelong learning leverages transfer learning in order to reduce the learning complexity of subsequent learning tasks. The lifelong learning problem can be framed as bias learning: the learner's task is to find a bias as an hypothesis space that is appropriate for the environment considered [Baxter, 2000]. As more tasks are observed, the system changes its bias to accommodate to the distribution of tasks presented. Lifelong learning has initially been introduced in robotics [Thrun and Mitchell, 1995].

The symbolic representation makes ILP systems naturally suited for lifelong learning. Since hypotheses are encoded in the same language as the background knowledge, learned hypotheses can easily be stored in the background knowledge [Muggleton *et al.*, 2012; Cropper *et al.*, 2020a]. Lifelong learning allows for ILP systems to acquire background knowledge through time instead of relying on predefined and manually chosen background knowledge. This is a form of bias shift.

For instance, Dependent Learning [Lin *et al.*, 2014] allows the automatic construction of a series of predicates with increasing levels of abstraction. The learner is given a series of tasks with varying complexity. The learner starts with a low limited complexity k and finds all tasks which can be solved using at most k clauses. Each learned definition is added in the background knowledge and the corresponding task is marked as solved. The clause bound k is incremented. Next, the learner attempts to solve remaining unsolved tasks while being allowed to reuse definitions learned in the previous depth bounds. This process repeats until

the maximum clause bound is reached or until all tasks are solved. Thus, Dependent Learning can automatically identify easier problems, solve them and reuse the solutions to help solve more complex problems [Cropper *et al.*, 2020a]. At the end, the learned program is a hierarchy of concepts: the system starts by learning simpler reusable rules, and builds more complex rules on top of each other. Dependent Learning has been demonstrated to improve learning performance over a series of tasks. Playgol [Cropper, 2019] acquires reusable knowledge in an initial unsupervised stage. Before solving the tasks at hand, the learner first plays: it randomly samples a set of tasks which it attempts to solve with Dependent Learning. Tasks are generated randomly by sampling instances from the instance space. Any learned solution is saved in the background knowledge as new invented predicates. After playing, Playgol solves the tasks at hand, while being allowed to reuse any solution learned whilst playing. In Chapter 6, we use a variant of Dependent Learning to learn game strategies.

2.3.9 Comprehensibility

The definition of Machine Learning presented in Definition 2.1 relies on a performance measure to evaluate the quality of Machine Learning systems. Michie defined Machine Learning performance in terms of two orthogonal axes: predictive accuracy and comprehensibility of generated hypotheses [Michie, 1988]. While early Machine Learning systems have only focused on predictive power, the need and importance for comprehensibility of learning systems has recently been acknowledged [Adadi and Berrada, 2018; Gunning and Aha, 2019]. Comprehensibility is fundamental in the context of many applications in which transparency and trust are essential. However, lack of comprehensibility is a limitation of current approaches which generally lack the ability to explain their decision and suffer from opacity. For instance, deep learning systems are criticised for being far not sufficiently transparent [Marcus, 2018]. Conversely, symbolic systems are inherently interpretable. In ILP, hypotheses are output as logic programs and usually are thought to be relatively easy to comprehend since verbal explanations can be generated from the reasoning trace in a straightforward manner [Muggleton *et al.*, 2012; Schmid, 2021].

A difficulty is to measure the comprehensibility and performance of learning systems. Michie specified three criteria for evaluating qualities of Machine Learning systems [Michie, 1988]. First, the *weak* criterion holds for machine learners which predictive performance improves with increasing amounts of data. Second, the *strong* criterion additionally requires the learning system to provide its hypotheses in symbolic form. Last, the *ultra-strong* criterion extends the strong criterion and additionally requires the machine learner to be able to teach the learned hypothesis to a human, whose performance is consequently increased to a level beyond that of the human studying the training data alone [Muggleton *et al.*, 2018b]. Recent work [Muggleton *et al.*, 2018b] provides an operational definition for comprehensibility of logic programs based on Michie's ultra-strong criterion. Comprehensibility is estimated using human participant trials. The authors evaluate the comprehensibility of ILP hypotheses and provide the first demonstration of Ultra-Strong Machine Learning for an ILP system. Results are based on experimental evidence of improvement of human performance after being presented with first-order machine learned logic theories. Results indicate that comprehensibility is affected not only by the complexity of the presented program but also by the existence of anonymous predicate symbols [Schmid *et al.*, 2017]. We introduce in Chapter 6 a purely symbolic learner and show the underlying learned model provides some form of comprehensibility which will be discussed.

2.4 Summary

We have reviewed in this Chapter related work in Computability, Machine Learning, Logic Programming and ILP. Specifically, we have reviewed related work in active learning and reinforcement learning for revising the instance space. We have characterised induction as the search for consistent hypotheses. We have discussed the role of the inductive bias to guide this search and thus to improve learning efficiency. We have surveyed related work on ILP, which is a form of machine learning for inducing hypotheses, as logic programs, from examples. We have examined the strengths of ILP. Specifically, we have mentioned features of MIL, a state-of-the-art ILP technique on which we will focus in the this thesis. The next Chapter formalises the MIL theoretical framework used throughout this thesis.

Chapter 3

Theoretical Framework

This Chapter introduces the logical notation and the MIL theoretical framework used throughout this thesis.

3.1 Logic Programming

We start by restating relevant logic programming terminology and refer the reader to [Lloyd, 1987; Nienhuys-Cheng, 1997] for a more detailed description of logic programming notation.

3.1.1 First-order syntax and semantics

A variable is first-order if it can be bound to a constant symbol or another first-order variable. A variable is denoted by a string of characters or digits. A function symbol or a predicate symbol is denoted by a string of characters or digits starting with a lower-case letter. The arity of a function or predicate is the number of arguments it takes. A function or predicate p with arity a is denoted as p/a. Function and predicate are said to be monadic or dyadic when they have arity one or two respectively. A constant symbol is a function or a predicate symbol with arity zero. The constant signature $\mathcal{C}$ is the set of all constants. The predicate signature $\mathcal{P}$ is the set of all predicate symbols. The alphabet consists of logical symbols (negation $\neg$, conjunction

$\wedge$, disjunction $\vee$, implication $\rightarrow$, equivalence $\leftrightarrow$, existential quantifier $\exists$ and universal quantifier $\forall$), the constant signature $\mathcal{C}$, the predicate signature $\mathcal{P}$, the set of function symbols and the variables. A language L is the set of all formulae that can be constructed from the symbols in the alphabet. A well-formed formula, or formula, is a finite sequence of symbols that is part of the language. A first-order term is a first-order variable, a constant symbol or a function symbol of arity n applied to a n-tuple of first-order terms. A term is ground if it contains no variables. A first-order atom is a predicate symbol of arity n applied to a n-tuple of first-order terms. An atom is ground if all of its terms are ground. A first-order literal is a first-order atom A or its negation $\neg A$, the former being called a positive literal and the latter a negative literal. A variable is first-order if it is quantified over terms. First-order variables are represented with lower-case letters (eg: a, b, c). The process of replacing existential first-order variables with new unique constants is called Skolemisation. The unique constants are called Skolem constants.

A clause or a rule is a disjunction of literals. First-order variables in a clause are existentially or universally quantified but quantifiers may be omitted for brevity. A clause is ground if it contains no variables. The head and the body of a clause are its set of positive and negative literals respectively. A Horn clause is a clause which contains at most one positive literal. A definite clause is a Horn clause which contains exactly one positive literal. A goal is a Horn clause which contains no positive literals. A Horn clause is unit if it contains exactly one positive literal and no negative literal. A fact is a ground unit clause. A clausal theory is a conjunction of clauses. A clausal theory in which all predicates have arity at most one or two is called monadic or dyadic respectively. A logic program is a finite clausal theory. A definite program is a logic program in which each clause is definite. A Datalog program is a first-order definite logic program which contains no function symbols other than constants, for which each variable in the head also appears in a non-negated first-order body literal and with stratification restrictions on the use of negation and recursion. In the following, we represent abstract representations of logic programs following logical conventions and executional logic programs as Prolog programs. Logic programs are named with calligraphic upper-case letters (eg: $\mathcal{P}$, $\mathcal{Q}$) apart from background knowledge programs and hypothesis programs which are represented with the usual ILP notation B and H respectively.

A first-order substitution θ is a finite set of the form: $\theta = \{x_1/t_1, ..., x_n/t_n\}$ where $n \geq 0$, the x_i are distinct first-order variables and the t_i are first-order terms. The substitution θ is a ground substitution if every t_i is ground. The instance of a formula F by a first-order substitution θ, called $F\theta$, is the formula obtained by simultaneously replacing each occurrence of the first-order variable x_i in F by the term t_i for all $1 \leq i \leq n$. A unifier for the formulae F_1 and F_2 is a first-order substitution θ such that $F_1\theta = F_2\theta$. F_1 and F_2 are said to be unifiable if there exists a unifier for F_1 and F_2. A most general unifier for the formulae F_1 and F_2 is a unifier θ for F_1 and F_2 such that, for any unifier σ for F_1 and F_2, there exists a first-order substitution γ such that $\sigma = \theta\gamma$. The clause C is said to θ-subsume D, or simply subsume D, denoted by $C \succeq D$, if there exists a first-order substitution θ such that $C\theta \subseteq D$ [Plotkin, 1969]. A clause C is more specific than a clause D if D subsumes C. Conversely, a clause C is more general than a clause D if C subsumes D. θ-subsumption is reflexive, transitive but not anti-symmetric. Subsumption is used to define a quasi-order over hypotheses spaces.

The Herbrand Universe $\mathcal{U}_L$ of a language L is the set of all ground first-order terms which can be formed out of the constants and function symbols appearing in L. The Herbrand Base $\mathcal{B}_L$ of a language L is the set of all ground atoms which can be formed out of the predicate signature $\mathcal{P}$ of L and the Herbrand Universe $\mathcal{U}_L$ of L. An interpretation I_D over a domain D is an assignment of truth values to the elements of the set D. An interpretation I_D is represented as a subset of the domain D which includes elements that are true in I_D. A Herbrand interpretation over a language L is an interpretation for L based on the Herbrand base. A model of a set of formulae F is an interpretation I over F in which every formula of F is true with respect to I. A formula is valid if every interpretation is a model, satisfiable if it has at least one model, unsatisfiable if it has no model. A Herbrand model of a set of formulae F is a Herbrand interpretation which is a model for F. A Herbrand model is minimal if no proper subset of it is also a model. For a definite first-order program, the intersection of all Herbrand models is the unique minimal Herbrand model called the least Herbrand model. A formula c is a logical consequence of a set of formulae F if every model of F is a model of c, it is written $F \models c$. For a definite first-order program $\mathcal{P}$, the set of all logical consequences of $\mathcal{P}$ is equal to the least Herbrand model of $\mathcal{P}$ [Van Emden and Kowalski, 1976]. We then define the immediate consequence operator of a

definite first-order program:

Definition 3.1 (Immediate Consequence Operator of a Definite First-Order Program)**.** *Let $\mathcal{P}$ be a definite first-order logic program. The immediate consequence operator $T_\mathcal{P}$ associated with $\mathcal{P}$ is a mapping from subsets of the Herbrand base $\mathcal{B}_\mathcal{P}$ to subsets of $\mathcal{B}_\mathcal{P}$ defined as:*

$$\forall I \subseteq \mathcal{B}_\mathcal{P}, T_\mathcal{P}(I) = \{\alpha \in \mathcal{B}_\mathcal{P} \mid \alpha \leftarrow B_1, ..., B_m, m \geq 0 \text{ is}$$
$$\text{a ground instance of a clause in } \mathcal{P} \text{ and } \{B_1, ..., B_m\} \subseteq I\}$$

For a definite program $\mathcal{P}$, the least fixed point of the immediate consequence operator $T_\mathcal{P}$ is equal to the least Herbrand model of $\mathcal{P}$ [Van Emden and Kowalski, 1976].

SLD-Resolution [Kowalski and Kuehner, 1971] is an inference rule for Horn clauses. Let $G_1 = \leftarrow A_1, ..., A_m, ..., A_k$ be a goal and $C = A \leftarrow B_1, ..., B_q$ be a first-order Horn clause. Then the goal G_2 derived by SLD-Resolution from G_1 and C is $\leftarrow A_1, ..., B_1, ..., B_q, ..., A_k$ when θ is a most general unifier of A_m and A. The atom A_m is called the selected atom in G_1 and G_2 is called a resolvent of G_1 and C. A SLD-Derivation of a program $\mathcal{P}$ and a definite goal G_0 is a possibly infinite sequence of goals $\{G_0, G_1, ...\}$ such that each G_{i+1} is the resolvent of G_i with some clause in $\mathcal{P}$. A finite SLD-Derivation of the empty clause from $\mathcal{P}$ is called a SLD-Refutation of $\mathcal{P}$. A SLD-tree of a goal G and a first-order program $\mathcal{P}$ is a tree which root is G and containing all possible derivations of G with $\mathcal{P}$. We then define the success set of a first-order logic program:

Definition 3.2 (Success Set of a First-Order Program)**.** *Let $\mathcal{P}$ be a logic program. The success set $SS(\mathcal{P})$ of a program $\mathcal{P}$ is the set of all atoms from the Herbrand Base $\mathcal{B}_\mathcal{P}$ with a refutation:*

$$SS(\mathcal{P}) = \{A \in \mathcal{B}_\mathcal{P} \mid \mathcal{P} \cup \neg A \text{ has an SLD-refutation } \}$$

For a definite first-order program, the success set is equal to its least Herbrand model [Apt and Van Emden, 1982].

A goal is decidable if there exists a terminating effective procedure for determining its truth value. Two desirable properties of logical systems are soundness and completeness. A system is

said to be sound if it allows only logical consequences of the premises to be derived. A system is said to be complete if every formula that is a logical consequence of the premises can be derived by this system.

In the following, we denote by $\mathcal{X}$ the instance space, which is the set of all possible examples and we denote by $\mathcal{H}$ the hypothesis space, which is the set of logic program definitions defined over the instance space and expressible with the hypothesis language.

3.1.2 Second-order syntax and semantics

We consider an extension of the first-order syntax and semantics to second-order logic. A variable is second-order, or higher-order, if it is quantified over predicate symbols. Second-order variables are denoted with upper-case letters (eg: P, Q, R). A second-order term is a second-order variable or a function symbol of arity n applied to a n-tuple of terms, at least one of them being second-order. An second-order atom is a predicate symbol of arity n applied to a n-tuple of terms, at least one of them being second-order. A second-order literal is a second-order atom A or its negation $\neg A$, the former being called a positive literal and the latter a negative literal. The process of replacing existential second-order variables with new unique constants is called Skolemisation. The unique constants are called Skolem constants.

A clause is second-order if it contains at least one second-order literal. Second-order variables in a clause are existentially or universally quantified but quantifiers may be omitted for brevity. A logic program is second-order if it contains at least one second-order Horn clause.

A second-order substitution θ is a finite set of the form: $\theta = \{x_1/t_1, ..., x_n/t_n\}$ where $n \geq 0$, the x_i are distinct variables, and the t_i are terms, and at least of the x_i, t_i is second-order. The instance of a formula F by a second-order substitution θ, called $F\theta$, is the formula obtained by simultaneously replacing each occurrence of the second-order variable x_i in F by the term t_i for all $1 \leq i \leq n$.

3.2 Inductive Logic Programming

Three settings have been defined for ILP differing by the type of training examples [De Raedt, 1997]. In the learning from interpretations setting [De Raedt and Džeroski, 1994; Blockeel *et al.*, 1999], an example is a set of facts representing a logical interpretation I. An example I is covered by an hypothesis H if I is a model for H. In the learning from proof setting [Passerini *et al.*, 2006], an example is a proof or a trace, and an hypothesis H covers an example P if P is a proof for H. We focus in the following on the learning from entailment setting [Muggleton and De Raedt, 1994]. When learning from entailment, training examples are clauses and the ILP problem in the learning from entailment setting is defined as follows.

Definition 3.3 (ILP Problem [Nienhuys-Cheng, 1997])**.** *Assume input is a pair (B, E) where B is a logic program representing the background knowledge, and $E = E^+ \cup E^-$ is a pair of sets of clauses representing positive and negative examples. The ILP problem is to induce an hypothesis H as a logic program which, together with the background knowledge, entails all the positive examples and none of the negative examples:*

$$\forall e^+ \in E^+, B \cup H \models e^+$$
$$\forall e^- \in E^-, B \cup H \not\models e^-$$

Given background knowledge B, an hypothesis H is said to cover the example e if e is entailed by the hypothesis H, that is if e is true in all Herbrand models of $B \cup H$. A theory H is said to be complete if H, together with the background knowledge B, entails all positive examples. A theory H is said to be consistent if H, together with the background knowledge B, does not entail any of the negative examples. H is said to be correct if it is both complete and consistent [Nienhuys-Cheng, 1997]. Definition 3.3 may be relaxed to cope with noise, in which case the ILP problem aims at finding an hypothesis H that covers as many positive examples as possible but as few negative examples as possible.

Although Definition 3.3 allows examples, background knowledge and hypotheses to be any well-formed formulae [Muggleton and De Raedt, 1994], we consider in this thesis that the examples E

are pairs of sets of ground atoms and that the background knowledge B is a definite program. We also restrict hypotheses H to be definite programs with no function symbol, ie Datalog programs. In Chapter 6, we relax this restriction and learn programs with stratified negation.

3.3 Meta-Interpretive Learning

MIL is a form of ILP on which we will focus in the following. We describe within this section the MIL framework which we will use throughout this thesis.

3.3.1 MIL Problem

MIL [Muggleton *et al.*, 2014; Muggleton *et al.*, 2015] is a subfield of ILP. As an ILP learner, a MIL learner follows Definition 3.3. We focus on the learning from entailment setting [Muggleton and De Raedt, 1994] although variants of MIL have been discussed within the learning from interpretations setting [Muggleton *et al.*, 2015]. The MIL problem is defined as:

Definition 3.4 (MIL Problem)**.** *Assume input is a pair (B, E) where the background knowledge B is a second-order logic program $B = B_c \cup M$ composed of a definite first-order logic program background knowledge B_c and second-order meta-rules M and the examples $E = E^+ \cup E^-$ is a pair of sets of ground atoms representing positive and negative examples. The MIL problem is to induce an hypothesis H as a definite logic program which, together with the background knowledge, entails all the positive examples and none of the negative examples:*

$$\forall e^+ \in E^+, B \cup H \models e^+$$

$$\forall e^- \in E^-, B \cup H \not\models e^-$$

$$B \models H$$

In other words, the goal of a MIL learner is to find a hypothesis which, together with the background knowledge, entails all of the positive examples but none of the negative ones. The

background knowledge contains first-order clauses and second-order clauses, the meta-rules.

3.3.2 Meta-rules

Compared to a traditional ILP learner, a distinctive feature of MIL is the use of meta-rules M as part of the background knowledge B. Meta-rules are higher-order clauses with existentially quantified and universally quantified first-order and second-order variables:

Definition 3.5 (Meta-rule [Muggleton *et al.*, 2015])**.** *Given a set of constant symbols $\mathcal{C}$ and a set of predicate symbols $\mathcal{P}$, a meta-rule is a higher-order formula of the form:*

$$\exists\sigma\forall\tau, P(s_1, ..., s_m) \leftarrow Q_1(t_{1,1}, ..., t_{1,i_1}), ..., Q_k(t_{k,1}, ..., t_{k,i_k}).$$

where σ, τ are disjoints sets of variables such that $P, Q_1, ...Q_k \in \sigma \cup \tau \cup \mathcal{P}$ and $s_1, ..., s_m, t_{1,1}, ..., t_{k,i_k} \in \sigma \cup \tau \cup \mathcal{P} \cup \mathcal{C}$. In general, quantifiers are omitted for brevity and meta-rules are denoted more concisely as:

$$P(s_1, ..., s_m) \leftarrow Q_1(t_{1,1}, ..., t_{1,i_1}), ..., Q_k(t_{k,1}, ..., t_{k,i_k}).$$

In the following, meta-rules are treated as second-order definite program provided as background knowledge. They are resolved alongside the examples and the first-order background knowledge to derive an hypothesis. Meta-rules act as program templates describing the form of clauses permitted in hypothesised programs. In this sense, meta-rules define the hypothesis language and in turn the hypothesis space. The use of meta-rules clarifies the declarative bias being employed and characterise MIL as a declarative Machine Learning approach [De Raedt, 2012]. The logical formalisation of meta-rules provides the ability to reason about them [Cropper and Muggleton, 2014]. Examples of usual meta-rules are shown in Table 3.1. Meta-rules provide expressive structures which allow for the representation of complex theories. For instance, a MIL learner can learn recursive theories by making use of recursive meta-rules such as *tailrec* presented in Table 3.1.

Name	Meta-rule
identity	$P(a,b) \leftarrow Q(a,b).$
inverse	$P(a,b) \leftarrow Q(b,a).$
conj	$P(a) \leftarrow Q(a), R(a).$
postcon	$P(a,b) \leftarrow Q(a,b), R(b).$
precon	$P(a,b) \leftarrow Q(a), R(a,b).$
chain	$P(a,b) \leftarrow Q(a,c), R(c,b).$
tailrec	$P(a,b) \leftarrow Q(a,c), P(c,b).$
curry1	$P(a,b) \leftarrow Q(a,b,R).$

Table 3.1: Examples of usual meta-rules. The letters P, Q, and R denote existentially quantified variables and the letters a, b and c denote universally quantified variables. In the following, meta-rules are treated as second-order definite programs.

The use of meta-rules leads to a more constrained and effective search. However, an appropriate choice of meta-rules is crucial: too few meta-rules limits expressivity and might not allow for the construction of consistent hypotheses while too many hypotheses can lead to an expensive search. Therefore, the choice meta-rules realises a trade-off between expressivity and efficiency and an appropriate choice of meta-rules can improve the learning performance of a MIL learner [Cropper and Tourret, 2018].

3.3.3 Hypothesis Construction

Definition 3.4 states that hypotheses are the instantiation of meta-rules. Existentially quantified variables in meta-rules are unified with symbols from $\mathcal{P} \cup \mathcal{C}$. These higher-order substitutions for existentially quantified variables in the meta-rules are called meta-substitution:

Definition 3.6 (Meta-substitution)**.** *Let M be a meta-rule. A meta-substitution for M is a higher-order substitution for the existentially variables in M.*

Meta-substitutions can be used to reconstruct the learned hypothesis as a logic program by projecting the substitutions onto their corresponding meta-rules. This reconstitutes a first-order definite program which is an inductive generalisation of the examples. Owing to the existentially quantified variables in the meta-rules, the resulting first-order theories are strictly logical generalisation of the meta-rules employed.

Example 3.1 (Meta-Substitution)**.** *A board is a list of non-empty cells. A cell is an atom of the form* $c(Rank, File, Color)$*. We assume a background knowledge containing the primitives* piece/2 *and* white/1 *and the* postcon *meta-rule represented in Table 3.1. We consider a set of positive examples containing a single example* $E^+ = \{white\_piece([c(1, f, w)c(6, c, b)], c(1, f, w))\}$ *and an empty set of negative examples* $E^- = \{\}$*. Suppose a MIL learner aims to learn a concept* $white\_piece/2$ *extracting a white piece from a board given as first argument. It can construct the following meta-substitution for the* postcon *meta-rule:*

$$\{P/white\_piece, Q/piece, R/white\}$$

This meta-substitution can be translated into the following logic program by substituting it back onto the corresponding meta-rule postcon*:*

$$white\_piece(A, B) : -piece(A, B), white(B).$$

The construction of hypotheses as meta-substitutions is performed during the derivation of the examples from the background knowledge. A Prolog meta-interpreter attempts to construct a proof of each example represented as goals. A standard Prolog interpreter attempts to prove a goal by repeatedly fetching first-order clauses whose heads unify with the given goal, and then proving first-order body literals. Conversely, a meta-interpreter additionally can prove a goal by fetching meta-rules whose heads unify with the given goal, before proving second-order body literals. The meta-interpreter saves meta-substitutions generated throughout successful proofs of the positive examples such that they can be reused in later proofs. After proving all goals, an hypothesised program is formed from the meta-substitutions built. These saved meta-substitutions are applied to their associated meta-rules to retrieve the learned hypothesis.

3.3.4 Predicate Invention

A characteristic feature of MIL is the support of predicate invention. Predicate invention is the automatic introduction of new auxiliary predicates which are not part of the initial

Background Knowledge B	Positive Examples E^+	Negative Examples E^-
mother(i,a).	grandparent(i,b).	grandparent(a,b).
mother(c,f).	grandparent(i,c).	grandparent(b,c).
mother(c,g).	grandparent(a,d).	grandparent(c,d).
mother(f,h).	grandparent(a,e).	grandparent(d,e).
father(a,b).	grandparent(a,f).	grandparent(e,f).
father(a,c).	grandparent(a,g).	grandparent(f,g).
father(b,d).	grandparent(c,h).	grandparent(g,h).
father(b,e).		grandparent(h,i).

Table 3.2: Learning the grandparent relation: first-order background knowledge and examples

predicate signature of $B \cup E$. Predicate invention extends the learner vocabulary $\mathcal{P}$. Within MIL, predicate invention is conducted by the meta-interpreter during the construction of meta-substitutions. The meta-interpreter automatically introduces a finite number of new higher-order Skolem constants representing new unique predicate names. These Skolem constants can be bounded to second-order variables in the meta-rules during the proof of the positive examples. The meta-interpreter then constructs a definition for these otherwise uninterpreted symbols.

Example 3.2 (Invention [Cropper and Muggleton, 2016])**.** *Consider the set of examples and the background knowledge containing the description of relations* $father/2$ *and* $mother/2$ *shown in Table 3.2 and the* chain *and* identity *meta-rules represented in Table 3.1. A MIL learner can learn the logic program below for describing the relation* $grandparent/2$*:*

$$grandparent(A, B) \leftarrow grandparent\_1(A, C), grandparent\_1(C, B).$$
$$grandparent\_1(A, B) \leftarrow mother(A, B).$$
$$grandparent\_1(A, B) \leftarrow father(A, B).$$

The predicate $grandparent\_1/2$ *is an invented predicate: it does not belong to the predicate signature of* $B \cup E$ *but appears in the learned hypothesis. This invented predicate has been introduced by the meta-interpreter as a new Skolem constant, and has been bound to second-order variables in the* chain *meta-rule. Its definition has been constructed with meta-interpretation as the disjunction of the predicates* $mother/2$ *and* $father/2$*. This invented predicate* $grandparent\_1/2$ *represents the parent relation.*

3.3.5 Higher-order programs

MIL systems not only can learn first-order programs but also support learning of higher-order programs. A higher-order program is a program that allows for quantification over higher-order variables that can be bound to predicate symbols.[1] Higher-order programs are learned by making use of higher-order definitions provided as background knowledge:

Definition 3.7 (Higher-order definition [Cropper *et al.*, 2020b])**.** *A higher-order definition is a set of higher-order Horn clauses where the head atoms have the same predicate symbol.*

All variables in a higher-order definition are universally quantified. However, by opposition with first-order definitions, variables may be first-order or second-order and are universally quantified over the set of constant symbols $\mathcal{C}$ or over the set of predicate symbols $\mathcal{P}$:

Example 3.3 (Higher-order definition)**.** *The definition* until *below is a higher-order definition:*

$$until(A, B, Cond, F) \leftarrow F(A, B), Cond(B).$$
$$until(A, B, Cond, F) \leftarrow F(A, C), until(C, B, Cond, F).$$

Whereas the variables $Cond$ and F are universally quantified higher-order variables, the other variables A, B, C are universally quantified first-order variables. This higher-order definition until/4 represents a recursive call to the action F and terminates when the condition $Cond$ is fulfilled.

An abstraction is a higher-order Horn clause that contains at least one atom which takes a predicate symbol an argument. An abstractions is the use of higher-order definitions to hide away the complexity of a program. The use of higher-order definitions provides simpler and more compact representations of the learned theories. Therefore, the use of higher-order definitions can reduce sample complexity and learning times and improve predictive accuracies [Cropper *et al.*, 2020b].

[1] In the following, we consider higher-order variables as second-order variables but we use the terminology higher-order for consistency with the literature and to indicate potential extension to higher-orders.

Example 3.4 (Abstraction)**.** *We consider a background knowledge containing the higher-order definition* until/4 *presented in Example 3.3, the predicate* move_up/2 *which increments the rank of a piece by one unit and* rank_8/1 *which holds if a piece is located on a position with rank 8. The following abstraction describes the concept of moving up until the eighth rank is reached for a pawn, in which case this pawn will be promoted. The last two arguments of* until/4 *in the abstraction below are ground to the predicate symbols* rank_8/1 *and* move_up/2*.*

$$promote(A, B) \leftarrow until(A, B, rank\_8, move\_up).$$

The use of the higher-order definition until/4 *abstracts away the manipulation of board states represented as lists. Recursion is implicit in the higher-order definition* until/4*. This avoids the need to learn an explicitly recursive program for defining* promote/2 *and provides a more compact definition for* promote/2*.*

Learning of higher-order programs can be combined with predicate invention to support higher-order predicate invention, which is the invention of predicates for use in higher-order constructs. For instance, a MIL learner can invent functions and conditions in the higher-order definition *until/4*. Meta-interpretation allows to alternate abstractions and invention to an arbitrary depth and thus can learn nested abstractions and inventions.

3.3.6 Search Space

We describe within this Subsection properties of MIL search spaces. We first restrict the hypothesis space to Datalog programs.

Proposition 3.1 (MIL decidable [Muggleton *et al.*, 2015])**.** *The MIL problem is decidable in the case M, B_c, E^+, E^- are Datalog and the predicate signature $\mathcal{P}$ and the constant signature $\mathcal{C}$ are finite.*

The restriction to Datalog program allows decidability when the predicate and constant signature are finite. However, the satisfiability of a MIL problem is undecidable when $\mathcal{C}$ is infinite.

We then define the restricted class of logic programs H^i_j which is a subset of Datalog:

Definition 3.8 (H^i_j program class [Muggleton *et al.*, 2015])**.** *For $i, j \in \mathbb{N}$, the language class H^i_j contains all definite Datalog programs constructed from the predicate signature $\mathcal{P}$ and the constant signature $\mathcal{C}$ with predicates of arity at most i and at most j atoms in the body of each clause.*

In this thesis, we mainly focus on the program class H^2_2, which consists of dyadic Datalog logic programs with arity at most 2 and with at most 2 atoms in the body of each clause. This restriction has several practical advantages. First, a restricted number of body literals limits the number of possible meta-rules. Moreover, this restriction forces more prolific predicate invention and the decomposition of hypotheses into reusable predicates. In addition, as stated by Proposition 3.1, this restriction allows for decidability: the satisfiability of a Datalog goal given a program belonging to the program class H^2_2 is decidable when the predicate signature $\mathcal{P}$ and the constant signature $\mathcal{C}$ are finite. It is however undecidable when $\mathcal{C}$ is infinite [Muggleton *et al.*, 2015]. Finally, the fragment H^2_2 still is highly expressive: it can be shown this fragment has Universal Turing Machine expressivity [Muggleton *et al.*, 2015].

The following proposition provides an upper bound over the number of programs in the program class H^i_j:

Proposition 3.2 (Size of the search space in H^i_j [Lin *et al.*, 2014; Cropper and Tourret, 2018])**.** *Assume n_{pred} predicate symbols and m meta-rules, and $i, j, n \in \mathbb{N}^*$. The number of H^i_j programs expressible with at most n clauses is $O(m^n n_{pred}^{(j+1)n})$.*

Proposition 3.2 shows that the size of the search space is exponential in the number of clauses n and polynomial in the number of meta-rules m and in the number of predicate symbols n_{pred}.

According to the Blumer bound [Blumer *et al.*, 1989] described in Section 2.1.3, the sample complexity is a function of the size of the search space. We can thus infer the following sample complexity result for the program class H^i_j:

Proposition 3.3 (Sample Complexity in H^i_j)**.** *Assume $\epsilon > 0$, $\delta > 0$, n_{pred} predicate symbols, m meta-rules, and $i, j, n \in \mathbb{N}^*$. The number of training examples n_{ex} required to PAC-learn*

with error at most ϵ and probability at least $1-\delta$ an hypothesis from the program class H^i_j with at most n clauses verifies:

$$n_{ex} \geq \frac{1}{\epsilon}(n\ln(m) + (j+1)n\ln(n_{pred}) + \ln(\frac{1}{\delta}) + ln(c)) \text{ with } c \text{ constant}$$

Proposition 3.3 shows that the sample complexity in the program class H^i_j is polynomial in the number of clauses and logarithmic in the number of meta-rules and in the number of predicate symbols. In this sense, selection of appropriate meta-rules [Cropper and Tourret, 2018] and predicate symbols [Cropper, 2020; Dumancic *et al.*, 2021] can improve the sample efficiency. We will see in Chapter 5 how the number of clauses in hypothesised programs can be reduced with predicate invention, which in turn can reduce the sample complexity.

3.3.7 *Metagol*

Several implementations of MIL have been described in the literature. The MIL problem has been encoded in an ASP problem for which ASP solvers attempt to find a model [Muggleton *et al.*, 2014; Kaminski *et al.*, 2018]. We focus in this thesis on a Prolog implementation called *Metagol* [Cropper and Muggleton, 2016].

To find an hypothesis, *Metagol* constructs a proof for the positive examples. It successively considers each positive example represented with a ground atom as a goal. For each goal, it first attempts to prove the atom considered using the background knowledge or clauses already induced. Failing this, it unifies the atom considered with the head of a meta-rule. The body of the meta-rule is converted to a set of atomic goals which are proved following the same process. Thereby, *Metagol* explores bindings of the existentially variables in this meta-rule to symbols from the predicate and constant signatures. The constructed meta-substitutions for any successful proof are saved as Prolog atoms. Thus, the meta-interpreter recursively proves a series of atomic goals by matching them against the background knowledge or the heads of available meta-rules. For any failure, *Metagol* backtracks and explores alternative choices. After proving all the positive examples, the learned hypothesis stored in the saved

meta-substitutions is complete by construction. This resulting hypothesis is then tested against the negative examples. If the hypothesis entails a negative example, *Metagol* backtracks to the last unexplored choice point during the proof of the positive examples and resumes the search. Otherwise, if no negative examples are covered, the hypothesis is returned. All other consistent hypotheses can be generated by further backtracking through the SLD-proof space: the search of *Metagol* is ordered via Prolog's procedural semantics. Completeness of SLD-resolution ensures that all hypotheses consistent with the examples can be constructed.

Metagol learns optimal hypotheses. It uses iterative deepening over the number of clauses to ensure the first hypothesis returned contains the minimal number of clauses. The search starts at depth 1. For each depth i, *Metagol* can build hypotheses with at most i clauses. If an hypothesis with at most i clauses which is consistent with the examples exists, it is returned. Otherwise *Metagol* continues to the next depth $i + 1$ until the maximum number of clauses allowed by the user is reached. In addition, the bound on the number of clauses puts a limit on the number of Skolem constants introduced which is the number of invented predicates allowed. At each depth i, *Metagol* augments $\mathcal{P}$ with up to $i - 1$ new predicate symbols which are named as extensions of the target predicate name [Muggleton *et al.*, 2015].

Metagol supports non-observable predicate learning [Moyle and Muggleton, 1997; Muggleton and Bryant, 2000; Law *et al.*, 2021], which is the induction of definitions for predicates other than the predicates represented in the examples. By opposition, in observational predicate learning, the examples and the target hypotheses define the same predicate. *Metagol* additionally supports multi-predicate learning which is the simultaneous learning of definitions of several predicates from a mixture of examples of these different predicates. These predicates may be inter-dependent. *Metagol* also can learn mutually recursive programs. For instance, it can learn the definition of an even number by inventing and learning the definition of an odd

number [Muggleton *et al.*, 2015]:

$$\text{even}(0).$$
$$\text{even}(A) : -\text{successor}(A, B), \text{even\_1}(B).$$
$$\text{even\_1}(A) : -\text{successor}(A, B), \text{even}(B).$$

Although *Metagol* supports both non-observable and multi-predicate learning, we focus in our experiments on observable and single predicate learning. We leave experimental evaluations of extensions to our contributions to these settings as future work.

3.4 Bayesian MIL

We describe within this section how MIL can be extended to implement a Bayesian posterior distribution over the hypothesis space.

3.4.1 Stochastic Refinement

A downward refinement operator is a function which maps a clause to a set of specialisations of this clause:

Definition 3.9 (Downward Refinement Operator [Shapiro, 1991; Nienhuys-Cheng, 1997])**.** *Let $\langle G, \succeq \rangle$ be a quasi-ordered set of clauses. A downward refinement operator is a function $\rho : G \to 2^G$, such that, for all $C \in G$ $\rho(C) \subseteq \{D \mid C \succeq D\}$. The set of one-step refinements ρ^1 and n-step refinements ρ^n of $C \in G$ are:*

$$\rho^1(C) = \rho(C)$$
$$\rho^n(C) = \{D \mid \exists E \in \rho^{n-1}(C) \text{ such that } D \in \rho(E)\} \text{ for } n \geq 2$$

The set of refinements ρ^* of $C \in G$ is the Kleene closure of the downward refinement operator

ρ applied to C:

$$\rho^*(C) = \rho^1(C) \cup \rho^2(C) \cup ...$$

A refinement operator induces a refinement graph: the refinement graph is a directed graph which has the members of G as nodes and which contains an edge from C to D if $D \in \rho(C)$. Moreover, according to Definition 3.9, the refinement of a clause is a set of clauses. Then the stochastic refinement of a clause is defined as a probability distribution over this set of clauses:

Definition 3.10 (Downward Stochastic Refinement Operator [Tamaddoni-Nezhad and Muggleton, 2011]). *Let $\langle G, \succeq \rangle$ be a quasi-ordered set of clauses and ρ downward refinement operator. A downward stochastic refinement operator is a function $\sigma : G \rightarrow 2^{G \times [0,1]}$ such that, for $C \in G$:*

$$\sigma(C) = \{\langle D_i, p_i \rangle \mid D_i \in \rho(C), p_i \in [0,1] \text{ and } \sum_{i=1}^{|\rho(C)|} p_i = 1\}$$

A σ-chain from $C \in G$ to $D \in G$ is a sequence $\{C_0, C_1, ...C_m\}$ with $C = C_0$ and $C_m = D$ such that for all $1 \leq i \leq m$, $\langle C_i, p_i \rangle \in \sigma(C_{i-1})$. The probability of this σ-chain is $\prod_{i=1}^{m} p_i$. Then, the n-step stochastic refinements of $C \in G$ is defined as:

$$\sigma^n(C) = \{\langle D_i, p_i \rangle \mid D_i \in \rho^n(C), p_i = \sum_{x \in X} p(x) \text{ where } X \text{ is the set of } \sigma\text{-chain from } C \text{ to } D_i\}$$

The stochastic refinements σ^* of a downward refinement operator ρ is defined as:

$$\sigma^*(C) = \{\langle D_i, p_i \rangle \mid D_i \in \rho^*(C), p_i \in [0,1] \text{ and } \sum_{i=1}^{|\rho^*(C)|} p_i = 1\}$$

The n-step stochastic refinements of a clause represent a probability distribution. This probability distribution can be viewed as a prior in a stochastic ILP search.

3.4.2 Bayes Theorem

In the context Bayesian MIL, we consider a downward refinement operator ρ which consists of the selection of a consistent meta-substitution followed by its abduction. Then, the downward stochastic refinement operator σ with respect to a meta-interpreter involves making selections according to a probability distribution over the meta-substitutions [Muggleton *et al.*, 2013]. In this case, the prior of an hypothesis H relative to the background knowledge B is defined from the stochastic refinements σ^* as:

$$p(H|B) = \sum_{\langle H,p\rangle \in \sigma^*(\neg B)} p$$

The likelihood of the examples E with respect to the background knowledge B and hypothesis H is evaluated as:

$$p(E \mid B, H) = \begin{cases} 1 & \text{if } B, H \models E \\ 0 & \text{else} \end{cases}$$

Bayes' theorem describes how a prior probability, which represents the initial degree of belief one has in a set of possible hypotheses, becomes a posterior probability as the result of observing some evidence:

$$p(H|B, E) = \frac{p(H \mid B)p(E \mid B, H)}{p(E \mid B)}$$

The denominator $p(E \mid B)$ does not rely on a choice of hypothesis and thus can be assimilated to a normalisation constant and be ignored when maximising the posterior.

Some learners usually are employed to make predictions given a posterior distribution. A Gibbs learner outputs a consistent hypothesis sampled according to the posterior distribution. A MAP (Maximum A Posteriori) learner outputs an hypothesis H with maximum posterior probability: $H \in \text{argmax}_H\, p(H \mid B, E)$. A Bayes learner classifies unseen instances as the averaged combination of the predictions of all consistent hypotheses weighted by their posterior

probability. A Bayes' prediction of instance x is defined as:

$$bayes(x) = \begin{cases} 1 & \text{if } \sum_H p(H \mid B, E) \geq 0.5 \\ 0 & \text{else} \end{cases}$$

A Bayes learner is an optimal prediction learner. A Gibbs learner has an expected error of at most twice that of a Bayes optimal learner [Haussler *et al.*, 1994]. In Bayesian MIL, stochastic refinement is used to randomly sample consistent hypotheses which are used to approximate Bayes' prediction.

3.4.3 MetaBayes

MetaBayes [Muggleton *et al.*, 2013] is a Bayesian MIL learner. It evaluates an approximation of Bayes' prediction based on averaging over the posterior probabilities of a set of sampled consistent hypotheses. MetaBayes samples consistent hypotheses using regular sampling. Regular sampling reproduces the effects of sampling without replacement, it limits the number of duplicates while maintaining a good sampling efficiency. Regular sampling works as follows. A set of probability fractions $\{f_1, ..., f_K\}$ is generated from the first K integers and with the following two properties: fractions generated are evenly distributed in $[0, 1]$ while containing no duplicates and the set of probability fractions is isomorphic to $\mathbb{N}$ for K infinite. We consider the derivation tree to be order left-to-right in SLD-order. Hypotheses are leaf nodes within this tree and each node corresponds to the choice of a meta-substitution. We consider the cumulative posterior probability of a leaf node to be the sum of posterior probabilities of hypotheses preceding it, that are hypotheses on its left in the tree. Samples are generated from this tree and following the probability fractions $\{f_1, ..., f_K\}$: for each fraction f_i, we return the rightmost hypothesis H_i such that it has cumulative posterior probability at most f_i: $\sum_{k=1}^{i} p(H_k|B, E) \leq f_i$. This hypothesis H_i is identified as follows. Branches are assigned probability intervals $[min, max]$ corresponding to the cumulative posterior probability associated with hypotheses found in the sub-tree under this branch. Starting at the root of the tree, the branch whose cumulative posterior probability interval $[min, max]$ verifies $min \leq f_i \leq max$ is chosen. Within the resulting

sub-tree, we repeat by updating f_i to $(f_i - min)(max - min)$ and selecting the sub-tree containing the updated probability f_i. This process is terminated when a leaf node is reached and the hypothesis corresponding to this leaf node is returned. At the end, regular sampling produces a sample set representative of the version space due to the evenly distributed sequence of fractions. MetaBayes supports sampling of hypotheses consistent with a given set of examples and background knowledge, and can be used to implement approximated Bayes' prediction.

3.5 Summary

In this Chapter, we have introduced the theoretical framework used throughout this thesis. We have used standard logical programming terminology. We have presented the MIL theoretical framework and its extension to Bayesian MIL. While the previous Chapters 2 and 3 have reviewed existing work, the following Chapters 4, 5 and 6 will introduce our contributions. The following Chapter presents an extension of Bayesian MIL with a method for efficiently building training sets with active learning for automated experiment selection.

Chapter 4

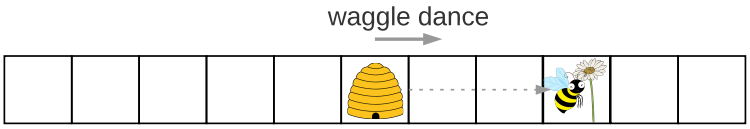
Active Bayesian Meta-Interpretive Learning

In this Chapter, we investigate Subthesis S.1 and introduce a method for revising the instance space in Bayesian MIL with active learning. This Chapter is based on the work published in [Hocquette and Muggleton, 2018].

4.1 Introduction

Once a honeybee has found a rich source of pollen, it shares its location with other members of the colony by executing a particular figure 8-movement called waggle dance. The bee orients its dance such that its direction indicates the location of the food source relative to the sun. Moreover, the duration of the waggle encodes the distance to flowers yielding nectar and pollen [Von Frisch, 1967]. This dance guides other bees and thus enhances the efficiency of the colony's foraging strategy.

More broadly, strategies are general programs aimed at achieving a goal, and that can provide plans for a multiplicity of initial states. We consider the task of learning agent strategies. However, when a scientist models animal behaviours or other strategies, the learning process generally requires the design and execution of many experiments. An experiment is the set-up



(a) First observation: the bee starts from the hive with no weight carried and ends up at the flower carrying pollen. This observation is labelled as positive.

Hypothesis 1	`f(A,B):-f_1(A,C),grab(C,B).` `f_1(A,B):-until(A,B,at_flower,move_right).`

Hypothesis 2	`f(A,B):-f_1(A,C),grab(C,B).`
	`f_1(A,B):-until(A,B,at_flower,f_2).`
	`f_2(A,B):-ifthenelse(A,B,waggle_east,move_right,move_left).`

(b) Two competing hypotheses consistent with the first observation (Figure 4.1a)

(c) Second experiment: it is discriminative between the competing hypotheses of Figure 4.1b, the knowledge of its label, no matter its value, will eliminate one of the competing hypotheses.

Figure 4.1: Observations of a bee behaviour

of an environment followed by the empirical observation of an outcome. Experiments allow for the arbitration of competing hypotheses and for knowledge acquisition. Experiments have an associated cost: their set-up and execution are resource exhausting and time-consuming. To that extent, learning efficiency relies on the number of experiments performed. We investigate in this work how much experimental cost can be reduced with active learning to learn agent strategies. An active learner is allowed to ask queries during the learning process. It chooses the next experiment to perform such that it is maximally discriminating between remaining competing hypotheses.

In Section 4.6, we learn a general strategy for a bee to find pollen in an environment. Learnt strategies are logic programs built from observations of bee behaviour. Observations are labelled as positive if the goal is fulfilled and as negative otherwise. Figure 4.1a represents a positive observation: the waggle dance indicates that a flower is at the right of the hive and the bee has successfully grabbed pollen thus the goal is fulfilled. Several hypotheses can be inferred from this observation, among them the two represented in Figure 4.1b. To discriminate be-

tween these two competing hypotheses, several experiments could be performed. One of these experiments is represented on Figure 4.1c: the flower now is on the left, which is indicated by the waggle dance. The first hypothesis predicts a negative outcome for this experiment while second one predicts a positive outcome. Therefore, the knowledge of the label of this experiment, regardless of its value, would eliminate one of these two hypotheses, which makes it an informative query.

MIL supports predicate invention, the learning of recursive programs [Muggleton *et al.*, 2014; Muggleton *et al.*, 2015] and the learning of higher-order programs [Cropper *et al.*, 2020b] thus is a suitable framework for learning agent strategies. In addition, in real-world situations, strategies should ideally be resource-efficient to be beneficial for agents. Therefore, we implent a prior which introduces a bias toward hypotheses with lowest complexity. More specifically, our framework is based upon Bayesian MIL, which is an extension of MIL that allocates a Bayesian prior distribution over the hypothesis space using stochastic refinement [Muggleton *et al.*, 2013]. To the best of our knowledge, Bayesian MIL is the only ILP framework that makes explicit distributional assumptions over the hypothesis space. Moreover, Bayesian MIL allows for efficient sampling of consistent competing hypotheses according to their posterior distribution. We extend Bayesian MIL into Active Bayesian MIL. Active Bayesian MIL additionally supports automated experiment selection based on active learning. We demonstrate Active Bayesian MIL can achieve lower sample complexity compared to Bayesian MIL.

Active Bayesian MIL works as follows. Given a set of labelled observations and some background knowledge, a set of consistent hypotheses are built. A Bayesian posterior distribution over the hypothesis space is defined from likelihood of the labelled observations and the hypotheses prior. The learner computes the entropy of possible experiments from the proportion of hypotheses weighted by their posterior that predicts a positive outcome for this experiment. An experiment with maximum entropy is selected: it is maximally discriminative between the remaining competing hypotheses and thus achieves the highest shrinkage of the version space. The learner observes the label of this experiment returned by an oracle. More iterations of this process are repeated during which more experiments are selected until convergence.

Specifically, the contributions of this Chapter are as follows. We introduce a framework, Active Bayesian MIL, which features automated experiment selection with active learning for learning efficient agent strategies with reduced cost of experimentation (Section 4.3). We theoretically compare the entropy of the instance selected between an active and a passive learner (Section 4.4). We provide and describe an implementation of the Active Bayesian MIL framework (Section 4.5). We experimentally demonstrate over two domains that Active Bayesian MIL converges faster toward agent efficient strategies than a passive learner in the same conditions (Section 4.6). Our experiments consider the task of learning bee strategies and learning deterministic Finite State Automata (FSA): our results demonstrate that the number of experiments to perform to reach an arbitrary accuracy level can at least be halved.

4.2 Related Work

4.2.1 Automated Scientific Discovery

Scientific discovery has been defined as the generation of novel, interesting, plausible, and intelligible knowledge about objects of study in science [Valdés-Pérez, 1999]. The automation of scientific discovery through the development of algorithms has long been of interest to accelerate scientific progress [King *et al.*, 2009; Gil *et al.*, 2014]. Early work focused on automating the generation of hypotheses identifying molecular structure to explain the data produced by a mass spectrometer [Buchanan *et al.*, 1969]. A later system [Langley *et al.*, 1987] can rediscover numeric scientific laws. *Prodigy* [Carbonell and Gil, 1990] implements a learning cycle for planning and problem solving. It refines its domain knowledge through experimentation.

In particular, ILP has been demonstrated suitable for designing systems capable of acquiring scientific knowledge. The Robot Scientist [Bryant *et al.*, 2001; King *et al.*, 2004; King *et al.*, 2009] is a closed loop logic-based Machine Learning system for fully automated Scientific Discovery and aimed at the determination of the function of yeast genes. It automatically originates hypotheses with ILP, devises experiments to test these hypotheses, executes these

experiments using an automated robotic system and interprets the results to falsify hypotheses inconsistent with the data observed. The Robot Scientist can autonomously propose and perform a sequence of experiments which reduces the expected cost of experimentation for converging upon an accurate hypothesis. However, this work was limited to finite hypothesis spaces and the demonstration in [King *et al.*, 2004] focuses on the abduction of single Prolog facts. We extend the class of learnable concepts to strategies involving predicate invention, recursion and higher-order programs in possibly infinite hypothesis spaces.

4.2.2 Active ILP

As described in Section 2.1.3, active learning is a framework in which the learner can query the label of unlabelled data of its choice to an oracle. An active learner aims to learn a model with high performance while minimising the cost of labelling data. While active learning was initially applied to classification tasks, active learning has been combined with ILP in several ways to learn more complex theories. For example, an active ILP learner has been designed for two non-classification tasks in natural language processing: semantic parsing and information extraction [Thompson *et al.*, 1999]. LOGAN-H [Arias *et al.*, 2007] learns first-order function-free Horn expression from interpretations by asking equivalence queries and membership queries. Active ILP has also been applied to identifying code search patterns and retrieving relevant code examples from corpus of data [Sivaraman *et al.*, 2019]. In the context of relational reinforcement learning, active exploration in relational worlds has been investigated [Lang *et al.*, 2010; Rodrigues *et al.*, 2011]. Conversely, this work integrates active learning with Bayesian MIL to devise a sequence of experiments for learning efficient agent strategies with reduced experimental costs.

4.2.3 Learning Grammars

The computational analysis of grammar learning has shown there is a polynomial time algorithm using both membership and equivalence queries [Angluin, 1987]. A polynomial time algorithm

for learning a subclass of Context-Free Grammars from positive examples of structural data has been described [Sakakibara, 1990]. However, by opposition with our approach, this algorithm cannot make use of negative experimental observations. Moreover, our approach is more general, it is not limited to grammars but can learn a broader class of hypotheses including agent strategies. MIL [Muggleton *et al.*, 2014] and Bayesian MIL [Muggleton *et al.*, 2013] have been successful at learning regular and context-free grammars from positive and negative examples due to their ability to support predicate invention and learning of recursion. However, these works did not featured automated experiment selection. Subsequent works have focussed on related language classes such as learning semantic conditions on Context-Free Grammars [Law *et al.*, 2019].

4.3 Theoretical Framework

We consider a probability distribution $D_{\mathcal{X}}$ over the instance space $\mathcal{X}$ and a probability distribution $D_{\mathcal{H}}$ over the hypothesis space $\mathcal{H}$. We assume that the target hypothesis H is drawn from $\mathcal{H}$ according to $D_{\mathcal{H}}$.

4.3.1 Complexity of an Hypothesis

Hypotheses from $\mathcal{H}$ differ by their complexity. The complexity of hypotheses $H \in \mathcal{H}$ usually is evaluated as the textual complexity $l(H)$ which measures the length of the logic program H. In the following, we evaluate the length of H as the number of clauses in H. The idea of learning minimal textual complexity hypotheses relies on Occam's principle described in Section 2.1.4. However, textually smaller programs are not necessarily the most desirable [Domingos, 1999]. Therefore, our framework considers an explicit prior distribution capable of incorporating domain knowledge. For instance, a desirable property of strategies is their efficiency. Therefore, we also evaluate the complexity of an hypothesis in terms of its resource complexity [Cropper and Muggleton, 2015]. The resource complexity $r(H)$ of an hypothesis $H \in \mathcal{H}$ is the sum of the action costs in applying the hypothesis H to the training examples. For a strategy, it is

the cost of transforming each initial training state into its corresponding final training state. Resource complexity can be viewed as a generalisation of time complexity, for which time is considered as a particular resource. In the following, we combine the textual complexity $l(H)$ and the resource complexity $r(h)$ into an overall complexity $c(H)$ defined as:

$$c(H) = l(H) + r(H) \tag{4.1}$$

4.3.2 Bayesian Prior Distribution

We consider the Bayesian MIL framework [Muggleton *et al.*, 2013] described in Section 3.4. Bayesian MIL implements a posterior distribution over the hypothesis space. This posterior is evaluated with Bayes' theorem from the prior and the likelihood. The likelihood is evaluated from the consistency of hypotheses with the training set. The prior is defined from the stochastic refinements with respect to the meta-interpreter and given the background knowledge B. The refinement operator consists of the selection of a consistent meta-substitution followed by its abduction. Our prior distribution encodes a preference for hypotheses with the lowest complexity. The complexity is defined as in Equation 4.1, from the number of stochastic refinements and the resource complexity. This prior results in a bias over the hypothesis space toward shortest and more efficient strategies. Our Bayesian prior probability is defined for any hypothesis $H \in \mathcal{H}$ as follows, $\frac{1}{a} = \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$ being a normalisation constant:

$$D_{\mathcal{H}}(\{H \mid c(H) = k\}) = \frac{a}{k^2} \tag{4.2}$$

4.3.3 Active Learning

A set of N instances is initially sampled from the instance space $\mathcal{X}$ according to $D_{\mathcal{X}}$. At each iteration $m > 0$, the active learner conducts an experiment in which it chooses the next instance x_m among this set and observes its label returned by an oracle. This information helps to discriminate between the current competing hypotheses since it rules out some proportion of

the version space V_m that is not consistent with it. We call $D_{\mathcal{H}}(V_m)$ the weight of the version space at the iteration m. We measure the shrinkage of the hypothesis space with the number $\frac{D_{\mathcal{H}}(V_{m+1})}{D_{\mathcal{H}}(V_m)}$ which represents the reduction ratio of the version space after the query of the label of the m^{th} instance. We associate each sampled instance x_m with a probability $p(x_m, V_m)$ given the version space V_m:

$$p(x_m, V_m) = \frac{\min(D_{\mathcal{H}}(\{H \in V_m \mid H(x_m) = 1\}), D_{\mathcal{H}}(\{H \in V_m \mid H(x_m) = 0\})}{D_{\mathcal{H}}(V_m)}$$

This probability represents the proportion of hypotheses which predict the minority outcome. By opposition with cover set approaches, the true label is unknown. This probability value represents the minimal reduction ratio over the version space V_m produced by the query of the instance x_m. We define the entropy of x_m as:

$$ent(p(x_m, V_m)) = -p(x_m, V_m)log(p(x_m, V_m)) - (1 - p(x_m, V_m))log(1 - p(x_m, V_m)) \quad (4.3)$$

From an information-theory point of view, the entropy of $p(x_m, V_m)$ is the expected information gain $\mathbf{E}_{H \backsim D_{\mathcal{H}}}[I(x_m, V_m, H)]$ following the knowledge of the label of x_m [Haussler *et al.*, 1994]:

$$\mathbf{E}_{H \backsim D_{\mathcal{H}}}[I(x_m, V_m, H)] = ent(p(x_m, V_m))$$

Instances with maximal entropy are maximally informative on expectation from the learner's point of view. These instances are the expected most discriminative given the current version space. Therefore, as noted in [Mitchell, 1978], an optimal query strategy is to select an instance covered by exactly half of the version space. In this case, the knowledge of its label, no matter its value, will halve the size of the version space. Therefore, the query strategy chosen is to select an instance x_m for which $p(x_m, V_m)$ is the closest to $\frac{1}{2}$, that is for which the entropy $ent(p(x_m, V_m))$ is maximal:

$$x_m = \underset{x}{\operatorname{argmax}}(ent(p_x, V_m))$$

Geometrically speaking, the $m+1$ first instances $X_m = \{x_0, ..., x_m\}$ generate a partition $T_{X_m, D_{\mathcal{H}}}$ of size at most 2^{m+1} over the hypothesis space. The expected cumulative information gain

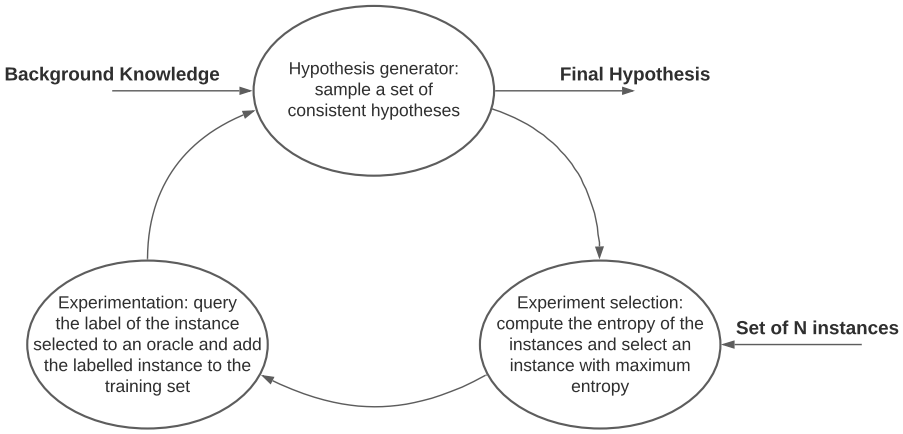


Figure 4.2: Diagram of Active Bayesian MIL's learning framework

$I(X_m, H)$ following the knowledge of labels of the instances $X_m = \{x_0, ..., x_m\}$ is the entropy of this partition weighted with the prior distribution:

$$\mathbf{E}_{H \backsim D_{\mathcal{H}}}[I(X_m, H)] = ent(T_{X_m, D_{\mathcal{H}}})$$

The entropy measures the uncertainty over the space of hypotheses. Thus, the information is maximised when the prior distribution tends to give equal weight to each element of the partition.

4.3.4 Learning Protocol

The learning protocol is summarised in Algorithm 4.1 and Figure 4.2. It represents how the learner acquires information. First, a set $S_{\mathcal{X}}$ of N instances is randomly sampled from the instance space $\mathcal{X}$ according to $D_{\mathcal{X}}$. The training set is initialised with one positive instance x_0 randomly selected from this set $S_{\mathcal{X}}$. This ensures that the learner can construct hypotheses since it can not learn from negative examples only. Then, a set $S_{\mathcal{H}}$ of K hypotheses consistent with the training set is sampled from $\mathcal{H}$ according to the posterior distribution. The entropy of each instance from $S_{\mathcal{X}}$ is computed from the set of sampled hypotheses $S_{\mathcal{H}}$. An instance with maximal entropy in $S_{\mathcal{X}}$ is selected. Its label is provided by the oracle $\mathcal{O}$ and this instance thus labelled is added to the training set. This process is repeated until the maximum number of iterations is reached, after which the hypothesis with the highest posterior is returned.

Algorithm 4.1 Active Bayesian MIL

Inputs: oracle $\mathcal{O}$, integers N, K and M, instance space $\mathcal{X}$ and hypothesis space $\mathcal{H}$, prior $D_{\mathcal{X}}$ and $D_{\mathcal{H}}$

Output: logic program hypothesis H

1: Sample a set $S_{\mathcal{X}}$ of N instances from $\mathcal{X}$ according to $D_{\mathcal{X}}$
2: Initialisation: randomly select a positive initial instance, set m=0
3: **while** the number of experiments m is lower than M **do**
4: Sample a set $S_{\mathcal{H}}$ of K hypotheses from $\mathcal{H}$ according to the posterior distribution
5: Compute the entropies of instances from $S_{\mathcal{X}}$ given $S_{\mathcal{H}}$
6: Select an instance x_m with maximal entropy from $S_{\mathcal{X}}$
7: Query the label of x_m to the oracle $\mathcal{O}$ and add it to the training set
8: m=m+1
9: **end while**
10: return the hypothesis H with the highest posterior from the sampled set $S_{\mathcal{H}}$

4.4 Theoretical Analysis

We theoretically evaluate within this section the instantaneous expected gain, that is the expected reduction of the version space V_m at some iteration m for a target class $\mathcal{H}$. As explained in the previous section, the expected information gain is the entropy of the instance selected. The entropy is computed from the probability $p(x_m, V_m)$ which represents the minimal reduction ratio.

Lemma 4.1 (Probability of selecting an instance with maximal entropy)**.** *A set of $N > 0$ instances $S_{\mathcal{X}}$ is randomly sampled from the instance space $\mathcal{X}$. The active learner selects an instance x_i with maximal entropy among this sample set $S_{\mathcal{X}}$. Then, the probability for an active learner of selecting an instance with maximal entropy is N times the one of a passive learner.*

Proof. The entropy is a monotonic function of the minimal reduction ratio. We consider an arbitrary distribution over $\mathcal{X}$ for the minimal reduction ratio. This distribution is bounded in $[0, \frac{1}{2}]$. We take $\frac{1}{N} > \epsilon > 0$ and p_ϵ is the probability value such that an ϵ-proportion of the instance space $\mathcal{X}$ has a reduction ratio greater or equal to p_ϵ. The instance selected by the active learner has a minimal reduction ratio $p_i < p_\epsilon$ if and only if every instance from the sample set has a minimal reduction ratio smaller than p_ϵ:

$$p(p_i < p_\epsilon) = p(p_1 < p_\epsilon, ..., p_N < p_\epsilon) = \prod_{k=1}^{N} p(p_i < p_\epsilon) = (1 - \epsilon)^N \tag{4.4}$$

Then, the probability for an active learner to select an instance with minimal reduction ratio at least p_i is:

$$p_{active}(p_i \geq p_\epsilon) = 1 - (1-\epsilon)^N = 1 - \sum_{k=0}^{N} \binom{N}{k} (-\epsilon)^k = N\epsilon + o(\epsilon) \tag{4.5}$$

By comparison, the probability for a passive learner to select an instance with minimal reduction ratio at least p_ϵ simply is:

$$p_{passive}(p_i \geq p_\epsilon) = \epsilon \tag{4.6}$$

Therefore, the probability of selecting an instance with maximal entropy is N times the one of a passive learner in the same conditions, where N is the number of unlabelled instances available. □

Lemma 4.1 is applicable to any active learner and is not restricted to ILP learners. However, in the following, our focus will be restricted to a MIL learner; our implementation and experiments are specific to MIL. We are interested in MIL for its ability to learn complex hypotheses which makes this framework suitable for learning agent strategies.

4.5 Implementation

4.5.1 Sampling a Set of Hypotheses

To cope with very large or potentially infinite hypothesis spaces, a set of consistent hypotheses is sampled at each iteration. This sample set is used both to approximate Bayes' prediction and to evaluate the entropies. Hypotheses are sampled with regular sampling [Muggleton *et al.*, 2013]. To ensure the sample set of hypotheses is not empty, we use dynamic sampling. A set of K hypotheses is sampled at each iteration, this sample size is doubled until at least one consistent hypothesis is found. First, a set of K unique and evenly distributed probability fractions is generated. For each fraction f_i, a sample hypothesis H_i is selected as a tree leaf and by following a path in the derivation tree and such that $\sum_{k=1}^{i} p(H_k \mid B, E) \leq f_i$. If H_i is

inconsistent with the training examples, it is discarded. If all sampled hypotheses have been discarded, a new set of hypotheses is sampled from the next K natural numbers, and so forth until at least one consistent hypothesis is returned. After removing potential duplicates, this sampled set of hypotheses is used to compute the entropies of sampled instances.

4.5.2 Computing the Entropies

In the next step, entropies are computed from the sampled hypotheses. For every candidate instance, we compute the proportion of sampled hypotheses weighted by the hypotheses prior that predict a positive label. The entropies are derived from this probability as per Equation 4.3. An instance with maximal entropy is selected, with ties broken at random.

4.6 Experiments

4.6.1 Experimental Hypothesis

This Section describes two experiments which evaluate the performance of Active Bayesian MIL over the speed of convergence when learning efficient agent strategies[1]. For the sake of comparison, we consider as baseline a passive learner which randomly selects one instance at each iteration. We investigate the following Experimental Hypothesis:

Experimental Hypothesis 4.1. *Active Bayesian MIL can converge to efficient agent strategies with a smaller sample complexity than Passive Bayesian MIL.*

We associate to the previous Experimental Hypothesis the following Null Hypothesis that we will test:

Null Hypothesis 4.1. *Active Bayesian MIL can not converge to efficient agent strategies with a smaller sample complexity than Passive Bayesian MIL.*

[1]The code for reproducing the experiments is available at `https://github.com/celinehocquette/Bayesian-MIL-active-learning.git`

Experiment	Name	Meta-rule
1	*acceptor*	$P(a,b) \leftarrow eq(a,b).$
	delta	$P(a,b) \leftarrow zero(a,c), Q(c,b).$ $P(a,b) \leftarrow one(a,c), Q(c,b).$
2	*chain*	$P(a,b) \leftarrow Q(a,c), R(c,b).$
	curry2	$P(a,b) \leftarrow Q(a,b,R,S).$
	curry3	$P(a,b) \leftarrow Q(a,b,R,S,T).$

Table 4.1: Meta-rules used in the experiments: the letters P,Q, R, S and T denote existentially quantified higher-order variables and the letters a, b, c universally quantified first-order variables.

```
one([1|T],T).
zero([0|T],T).
eq(A,A).
```

Table 4.2: Learning regular grammars: background knowledge

4.6.2 Material and Methods

Learning Regular Grammars The first experiment considers the task of learning regular grammars. Regular grammars are equivalent to FSA. Generally speaking, FSA represent sequences of actions depending on a sequence of events and an input state. Thus, they consist in compact ways of representing strategies. Our experimental material and methods are inspired from [Muggleton *et al.*, 2014]. We consider an alphabet $\Sigma = \{0, 1\}$. Let ν be a set of non-terminal symbols disjoint from Σ. We call λ the empty string. A grammar is a pair (s, R) consisting of a start symbol s and a finite set of production rules R. A grammar is regular if contains only production rules of the form $S \rightarrow \lambda$ or $S \rightarrow aB$ where $S, B \in \nu$ and $a \in \Sigma$. The meta-rules provided are *acceptor* and *delta* represented in Table 4.1, they represent these allowed form of production rules. As shown in Table 4.1, these meta-rules for regular grammars can be expressed with the *chain* and *identity* meta-rules only which are usual meta-rules represented in Table 3.1. The background knowledge is represented in Table 4.2a: these predicates parse letters from the alphabet. Target grammars are generated with *Metagol* from a set of sequences regularly sampled from Σ^*, half labelled as positive and half as negative. The number of states $n \geq 3$ is generated according to an exponential decay distribution with mean 4. A new number of states is generated following the same process to bound the search space for the learner. We add a constraint to *Metagol* to ensure target grammars are deterministic. We

```
q0(A,B):-zero(A,C),q1(C,B).
q0(A,B):-one(A,C),q0(C,B).
q1(A,B):-zero(A,C),q0(C,B).
q1(A,B):-one(A,C),q1(C,B).
q0(A,B):-eq(A,B).
```

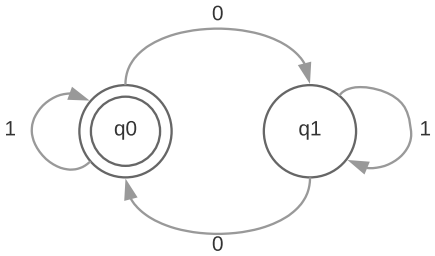


Figure 4.3: Example of target hypothesis: the parity grammar. q0 and q1 are non-terminal symbols and A,B,C are first-order variables.

additionally require target grammars to have a generality $g(H) = D_{\mathcal{X}}(\{x \in \mathcal{X} \mid H(x) = 1\})$ verifying $\frac{1}{3} < g(H) < \frac{2}{3}$ such that the initial probability for an instance to be positive is about $\frac{1}{2}$. This also ensures that trivial grammars are not considered. The generality of hypotheses is measured against a set of 40 new regularly sampled instances. These steps are repeated until a grammar with generality verifying $\frac{1}{3} < g(H) < \frac{2}{3}$ is found. An example of a target hypothesis and its corresponding automaton are represented on Figure 4.3. This example target hypothesis is the parity grammar: it accepts sequences with an even number of 0 and any number of 1. The complexity of an hypothesis is set to its length $l(H)$ measured as its number of clauses and the prior is computed as $\frac{1}{l(H)^2}$. For each run, a pool of 150 training instances are initially regularly sampled from Σ^*. A threshold on the probability fraction used for sampling is randomly generated for each instance, thus their length is bounded. Another 50 instances are similarly sampled for testing. A time-out is set to 10 minutes for each call to the hypothesis generator. A singleton set containing the empty hypothesis is output if no consistent hypotheses are sampled within the time-out. At each iteration, 50 hypotheses are regularly sampled. The accuracy is measured as the average accuracy of all consistent sampled hypotheses over the testing set. Results have been averaged over 50 trials.

Learning a Bee Strategy The second experiment considers the task of learning an agent strategy. The world is a one-dimensional space of size 10. The state of the world is described as a list of facts. Fluents are monadic predicates which verify conditions over a situation. The available fluents are *at_hive/1*, *at_flower/1* and *waggle_east/1*, they all have a cost of 0. Actions are dyadic predicates that modify the state of the world. The bee can perform

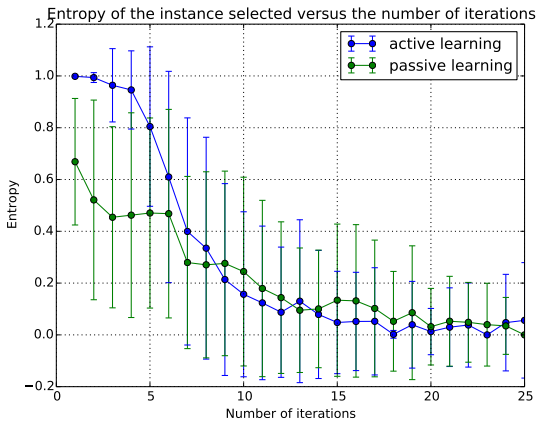
Name	Higher-order definition
until	$until(a,b,Cond,Func) \leftarrow Func(a,b),Cond(b).$ $until(a,b,Cond,Func) \leftarrow Func(a,c),until(c,b,Cond,Func).$
ifthenelse	$Ifthenelse(a,b,Cond,Then,Else) \leftarrow Cond(a),Then(a,b).$ $Ifthenelse(a,b,Cond,Then,Else) \leftarrow Else(a,c),eq(c,b).$

Table 4.3: Higher-order definitions used in the bee experiment. The symbols $Cond$, $Func$, $Then$, $Else$ denote existentially quantified higher-order variables. The letters a, b and c denote universally quantified first-order variables. The predicate *eq/2* holds when equality between its two arguments.

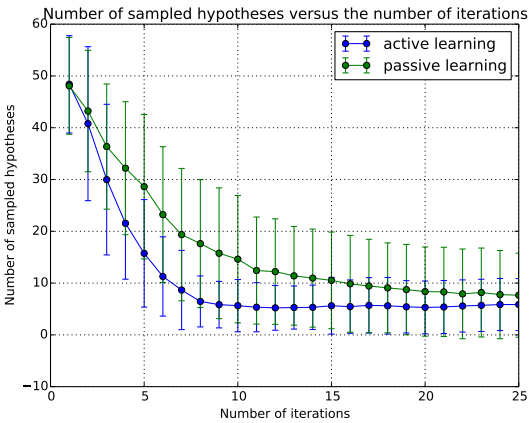
```
f(A,B):- f_1(A,C),grab(C,B).
f_1(A,B):- until(A,B,at_flower,f_2).
f_2(A,B):- ifthenelse(A,B,waggle_east,move_right,move_left).
```

Table 4.4: Target hypothesis for the bee experiment

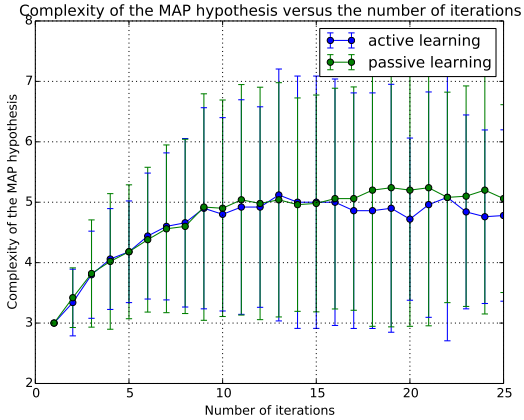
the following primitive actions: *move_right/2*, *move_left/2*, *grab/2*. Each of them costs one unit of energy. We allow the use of the two higher-order definitions *until/4* and *ifthenelse/5*. These higher-order definitions are represented in Table 4.3. The higher-order definition *until/4* repeatedly applies a dyadic action *Func* and terminates when a monadic condition *Cond* is fulfilled. The higher-order definition *ifthenelse/5* expresses a choice between the dyadic actions *Then* and *Else* based upon the realisation of the monadic condition *Cond*. These higher-order definitions can be expressed with variants of the *chain*, *precon* and *postcon* meta-rules only as shown in Table 4.3. The meta-rules provided to the learner in this experiment are presented in the Table 4.1. They are the usual *chain* meta-rule and the meta-rules *curry2* and *curry3*. These variants of the *curry* meta-rule are necessary to interpret higher-order definitions [Cropper *et al.*, 2020b]. For any hypothesis H with length $l(H)$, the resource complexity $r(H)$ is measured against the labelled examples, and the prior of H is then defined as $\frac{1}{l(H)+r(H)}$. A clause bound over the length of hypotheses is set to 3. The hive is located in the middle position of the environment. A flower is positioned in the environment and a waggle dance indicates whether it is east or west of the hive. In the initial state, the bee is at the hive with no pollen carried. It has some amount of energy randomly generated between 0 and 30. In the final state, it is on the flower with zero or one unit of pollen carried. Positive examples are pairs of states for which the task of finding pollen is fulfilled and with a positive amount of energy in the final state. Negative examples are pairs of states for which the task is not fulfilled or resulting in



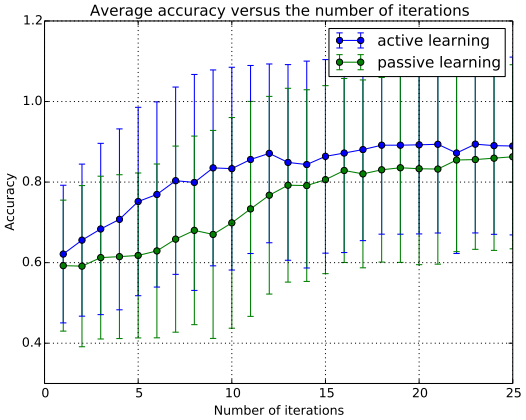
(a) Entropy of the instance selected versus the number of iterations



(b) Number of sampled hypotheses versus the number of iterations



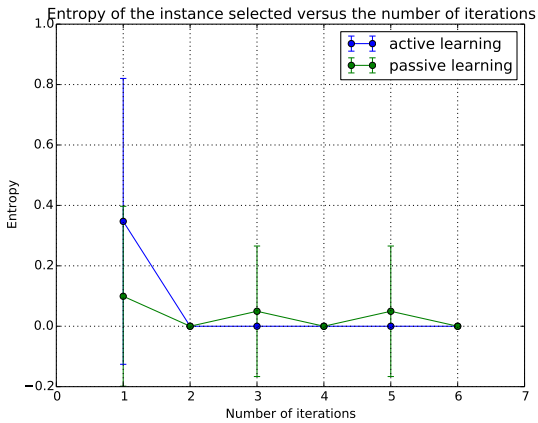
(c) Complexity of the sampled hypothesis with the highest prior versus the number of iterations



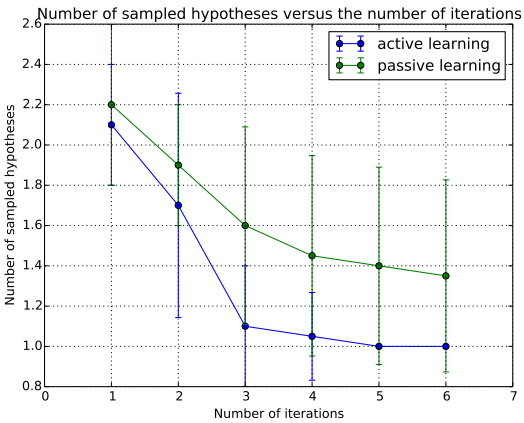
(d) Accuracy versus the number of iterations

Figure 4.4: Learning regular grammars with Active Bayesian MIL: comparison between active and passive learning; the convergence is faster for active learning.

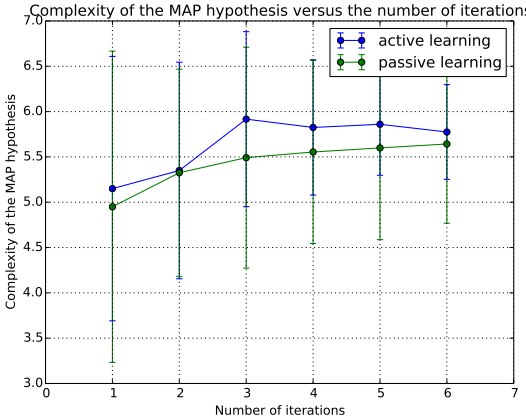
a strictly negative amount of energy in the final state. We learn the strategy introduced in Section 4.1 and represented in the Table 4.4. This strategy describes a honeybee behaviour for finding pollen in an environment, starting from the hive and following information given by a waggle dance. This strategy states that until a flower yielding pollen is reached, if the waggle dance indicates the location of a food source at the east of the hive, then the bee should move toward the right and otherwise it should move toward the left. Once at the flower, the bee grabs pollen. Pools of training instances and testing sets are respectively made of 20 and 40 examples, half positive and half negative. Results have been averaged over 20 trials. The hypothesis space being sparse, we sample 2000 hypotheses at each iteration.



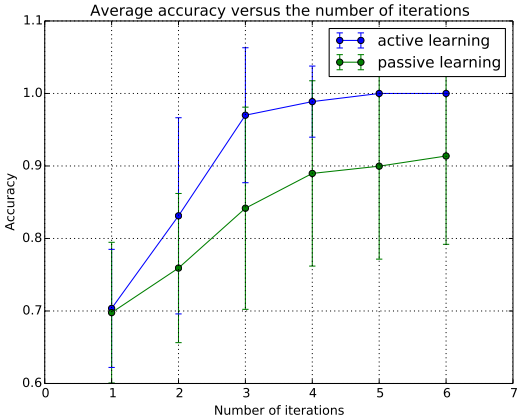
(a) Entropy of the instance selected versus the number of iterations



(b) Number of sampled hypotheses versus the number of iterations



(c) Complexity of the sampled hypothesis with the highest prior versus the number of iterations



(d) Accuracy versus the number of iterations

Figure 4.5: Learning a bee strategy with Active Bayesian MIL: comparison between active and passive learning; the convergence is faster for active learning.

4.6.3 Results

Results are presented on Figures 4.4 and 4.5. The number of iterations on the x-axis is the number of training examples queried. Learning time is around a few seconds per iteration for the bee experiment. The entropy of the instance selected is represented on Figures 4.4a and 4.5a. The entropy is smaller and less regular for passive learning and higher for active learning, especially for a small number of iterations. In both cases the entropy is globally

Accuracy	Active learning	Passive learning
0.75	5	12
0.80	7	15
0.85	11	22

(a) FSA

Accuracy	Active learning	Passive learning
0.80	2	3
0.90	3	6

(b) Bee experiment

Table 4.5: Number of iterations required to reach some accuracy levels for active and passive learning. These results suggest that experimental costs can be halved with active learning.

decreasing as the version space shrinks: it is less likely to find maximally informative instances in the sampled pool as the set of informative instances shrinks with the version space size. The difference between the two curves, blue and green, represents the gain over the reduction of the version space between active and passive learning. This difference weakly supports Lemma 4.1 although we would need to directly observe the probability of sampling instances with maximal entropy to experimentally verify Lemma 4.1.

The number of sampled hypotheses is represented on Figure 4.4b and 4.5b: it is decreasing with the number of iterations, eventually converging. The number of consistent hypotheses is smaller for active learning compared to passive learning. It represents the size of the version space, which shrinks faster for active learning since instances selected have higher entropy thus are more discriminative. In both cases, the decay rate gets smaller as the entropy drops. The number of hypotheses does not decrease by a factor of two as in the ideal case, even for active learning since the entropy of the instance selected generally is smaller than 1. Indeed, the existence of an instance with maximal entropy 1 is not guaranteed, and the selection of a high entropy instances depends on the size of the sampled pool as stated in Lemma 4.1.

The complexity of the MAP hypothesis (Figure 4.4c and 4.5c) increases with the number of iterations both for passive and active learning. Indeed, the posterior introduces a bias toward less complex hypotheses. Moreover, as more examples are available, more complex hypotheses need to be considered. The complexity and prior of the MAP hypotheses consequently are respectively increasing and decreasing as the number of iterations increases.

Finally, the average accuracy is represented in Figure 4.4d and 4.5d increases. The accuracy

converges toward 1 for the bee experiment. The accuracy converges below 1 for the grammar experiment since the number of sampled hypotheses consistent with the training set has not converged to 1. Some complex hypotheses may only be approximated with the number of states allocated. For both experiments, the default accuracy is around 0.5 and the learning process starts with one positive instance for initialisation which explains the initial accuracy between 0.6 and 0.7. In both experiments, fewer iterations are required to reach any given accuracy level with active learning compared to passive learning. Table 4.5 compares the number of queries required to reach some accuracy levels for active and passive learning: it suggests that experimental costs can at least be halved with active learning. A Mann-Whitney U test indicates that the results are significant at a 0.01 level, thus refuting our Null Hypothesis 4.1. In our framework, the entropy of a set of sampled instances is evaluated against sampled hypotheses. As demonstrated by Lemma 4.1, the number of instances sampled augments the probability of the existence of high entropy instances. In this sense, a larger sample size can augment the speed of convergence. Thus, the number of instances sampled is a parameter which involves a trade-off between computational cost and speed of convergence. The size of the set of hypotheses regularly sampled affects the degree of approximation of the version space and thus the precision on the entropy measured and on the evaluated accuracy.

4.7 Future Work

We identify the following limitations of these contributions which could be addressed as future work.

Theoretical Analysis A limitation of the contributions of this Chapter is the lack of theoretical bounds on the sample complexity, which we characterise as future work. These theoretical bounds will provide average case sample complexity for different class of hypothesis spaces and be expressed as a function of the number of hypotheses and instances sampled. These theoretical bounds over the sample complexity could be used to compare different active learning query strategies and to measure their respective benefits over passive learning for some target

hypothesis classes.

Cost Function In our current implementation, we have assumed a fixed cost for acquiring each label and we have not explicitly represented experimental costs in the form of a utility function. However, experimental cost may vary greatly over the instance space. For instance, labelling cost may be represented as a function of the length of sequences in the grammar experiment. Therefore, it would be valuable to further extend our framework by incorporating a cost function representing the effort needed for labelling an instance. This cost will represent the complexity of the realisation of an experiment and the amount of experimental resources it requires. This framework could also be extended to allow for limited experimental resources submitted to constraints. Finally, we also suggest extending this framework to the case in which the cost of labelling instances is unknown to the learner. For instance, the active learner can learn a predictive model of the unknown labelling cost alongside the task model [Settles *et al.*, 2008; Vijayanarasimhan and Grauman, 2009].

4.8 Summary

This Chapter has extended Bayesian MIL into Active Bayesian MIL and features automated experimentation with active learning for learning efficient agent strategies. This approach supports the idea of making efficient use of experimental costs while maintaining good predictive accuracies. We have introduced a framework for Active Bayesian MIL. A Bayesian posterior distribution is allocated over the hypothesis space. At each iteration, the learner queries the label of an instance with maximal entropy given the hypothesis space posterior. We have theoretically demonstrated that the probability of selecting an instance with maximal entropy is N times the one of a passive learner in the same conditions, where N is the number of unlabelled instances available. We have provided an implementation of Active Bayesian MIL. We have experimentally demonstrated over two domains that Active Bayesian MIL converges faster toward efficient strategies than a passive learner in the same conditions and that experimental costs can be halved with active learning compared to passive learning.

This Chapter has introduced a method to revise the instance space. We have demonstrated this method can reduce the sample complexity of MIL, thus supporting Subthesis S.1. Conversely, the next Chapter 5 introduces and evaluates a method to revise the hypothesis space to improve the sample and learning complexity of MIL and thus investigates Subthesis S.2. Both these Chapters will be extended in Chapter 6 with methods to revise both the instance and hypothesis space and thus will investigate Subthesis S.3. While this Chapter has focused on learning strategies for single agents, Chapter 6 will be adapted to learn strategies for describing the behaviour of an agent evolving in an adversarial environment, with applications to game playing.

Chapter 5

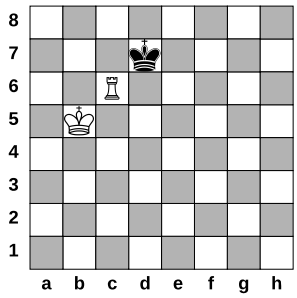
Complete Bottom-up Predicate Invention in MIL

In this Chapter, we investigate Subthesis S.2 and introduce a method for revising the hypothesis space using predicate invention. This Chapter is based on the work published in [Hocquette and Muggleton, 2020].

5.1 Introduction

We recall the example introduced in Section 1.1.4 in Chapter 1 and presented again in Figure 5.1. An optimal strategy in the chess endgame KRK is to confine the black king in an increasingly smaller area. This area is delimited by the rank and file controlled by the white rook. This white rook therefore is essential and must be maintained safe until checkmate. Threats against this white rook can be blocked using the white king. An example of a situation in which the white king protects the white rook is represented in Figure 5.1a. Figure 5.1b shows an hypothesis describing such a situation. This hypothesis is divided into several invented predicates representing sub-concepts.

We introduce a new method for partially delegating the construction of invented predicates and demonstrate it can improve learning performance. The hypothesis is split into substrate



(a) Example board.

Surface	`rp(S):-rp_1(S,P1),rp_3(S,P1).`

Substrate	`rp_1(S,P1):-rp_2(S,P1),white(P1).`
	`rp_2(S,P1):-piece(S,P1),king(P1).`
	`rp_3(S,P1):-rp_4(S,P2),distance1(P2,P1).`
	`rp_4(S,P2):-piece(S,P2),rook(P2).`

(b) Target Hypothesis: S denotes a board state, $P1$ and $P2$ are pieces

Figure 5.1: Learning a chess pattern: the white king protects its rook. It is black-to-move. See Figure 1.1 and Table 1.3.

and surface, as in Figure 5.1b. The substrate is a set of invented predicates generated in a first step by a bottom-up learner and from the background knowledge. The surface is an hypothesis built subsequently by a top-down learner which can reuse these substrate predicates. The use of substrate predicates results in a shorter surface hypothesis which thus is easier to learn.

In MIL [Muggleton *et al.*, 2015], predicate invention is performed during the search for a consistent hypothesis conducted top-down. New predicate symbols represented with Skolem constants are introduced in meta-substitutions. Conversely, we augment MIL systems with a new method to perform predicate invention in an initial pre-processing step. The availability of these invented predicates facilitates the subsequent search for a consistent hypothesis. Predicates are invented bottom-up from the background knowledge using a new method based on an extension of the immediate consequence operator for second-order logic programs. For each meta-rule, if body literals can be resolved with the current background knowledge related to the examples, then Skolem constants are bound to second-order variables in the head. The resulting head is added to the background knowledge and the resulting meta-substitution is saved as a new predicate definition. This process is iterated. We theoretically demonstrate our bottom-up method is complete with respect to a fragment of dyadic Datalog. Moreover, performing bottom-up iterations reduces the number of surface clauses to be learned by the

top-down learner which in turn can reduce the sample complexity. This new method provides a way to perform extensive predicate invention useful for feature discovery.

The contributions of this Chapter are as follows. We introduce a new method for performing extensive predicate invention (Section 5.3). We formalise the definition of an equivalence relation for predicates which is used to prune redundant predicates (Section 5.3). We provide a theoretical upper bound over the number of predicates generated (Section 5.4). We theoretically prove the completeness of this method with respect to a fragment of dyadic Datalog (Section 5.4). We theoretically prove our method reduces the number of surface clauses, which in turn can reduce the sample complexity (Section 5.4). We provide and describe an implementation of our method (Section 5.5). We demonstrate experimentally over two domains that this method can significantly improve learning performance (Section 5.6).

5.2 Related Work

5.2.1 Predicate Invention

In this subsection, we extend the general overview of predicate invention given in Section 2.3.5. Early predicate invention approaches were based on the use of W operators within the inverting resolution framework [Muggleton and Buntine, 1988]. However, the completeness of this approach was never demonstrated, partly because of the lack of a declarative bias to delimit the hypothesis space [Muggleton *et al.*, 2015]. Alternatively, predicate invention can be performed by adding new predicate symbols to mode declarations [Corapi *et al.*, 2011; Law, 2018; Evans and Grefenstette, 2018]. However their number and arity has to be user-provided while our method supports automated predicate invention. MIL systems [Muggleton *et al.*, 2014; Muggleton *et al.*, 2015] achieve predicate invention in a top-down fashion by substituting second-order variables in the meta-rules with higher-order Skolem constants representing new predicate symbols. The number of new constants introduced depends on the depth in the iterative deepening search. Conversely, we investigate bottom-up predicate invention in MIL and do not directly bound the number of invented predicates but only bound the depth of

the generation process. Finally, predicate invention can be performed as a form of meta-learning over time. Dependent Learning [Lin *et al.*, 2014] and Playgol [Cropper, 2019] allow the construction of a series of predicates with increasing levels of abstraction by solving a series of tasks with different complexity. In both cases, learned hypotheses are saved to the background knowledge as predicate definitions that can be reused when solving subsequent tasks. Both approaches are based on a set of tasks, user-provided or randomly sampled. Conversely, our method builds predicate definitions from the background knowledge related to the examples and does not require additional training tasks. Moreover, our method is complete.

5.2.2 Combining Top-down and Bottom-up approaches

We extend the review of hypothesis search methods from Section 2.3.7. Bidirectional hypothesis search strategy was originally presented in the version space algorithm [Mitchell, 1982]. Variants were implemented in algorithms alternating generalisation and specialisation steps to search through the lattice of clauses [Fensel and Wiese, 1993; Zelle *et al.*, 1994; Muggleton, 1995; Srinivasan, 2001; Califf and Mooney, 2003]. However, most of these systems [Fensel and Wiese, 1993; Muggleton, 1995; Srinivasan, 2001; Califf and Mooney, 2003] do not support predicate invention which restrict their expressivity. [Zelle *et al.*, 1994] includes a mechanism for demand driven predicate invention. Conversely, our system is based on MIL and fully supports automated predicate invention. Moreover, our bottom-up mechanism is not used during the search for a consistent hypothesis but prior to the search to shape the hypothesis space according to the learning task.

5.3 Learning Framework

5.3.1 Immediate Consequence Operator

We investigate in this work predicate invention in MIL. MIL systems make use of background knowledge treated as a second-order definite program. The background knowledge contains

first-order background knowledge and second-order clauses called meta-rules. As described in Section 3.3.4, predicate invention in MIL is usually performed by substituting second-order variables in meta-rules with Skolem constants representing new predicates. This is achieved top-down during the construction of a proof for the positive examples. Conversely, we present a method for generating predicates bottom-up from second-order background knowledge and in a pre-processing step. Predicates are constructed from an extension of the immediate consequence operator for second-order logic programs.

We first refer to the definition of the immediate consequence operator for first-order logic programs [Van Emden and Kowalski, 1976] stated in Definition 3.1. The immediate consequence operator $T_{\mathcal{P}}$ for first-order logic programs is a mapping from subsets of the Herbrand base to subsets of the Herbrand base. It derives the set of ground atomic logical consequences of a program given some interpretation. These logical consequences are identified as the head atoms of ground instance of clauses which body literals are true under the interpretation considered. The grounded variables are first-order variables which have been bounded to elements of the Herbrand Universe. MIL systems consider second-order logic programs background knowledge. Therefore, we extend Definition 3.1 to allow for second-order logic programs:

Definition 5.1 (Immediate Consequence Operator of a Definite Second-Order Program)**.** *Let $\mathcal{P}$ be a definite second-order logic program. The immediate consequence operator $T_{\mathcal{P}}$ associated with $\mathcal{P}$ is an operator defined over subsets of $\mathcal{B}_{\mathcal{P}}$ as:*

$$\forall I \subseteq \mathcal{B}_{\mathcal{P}}, T_{\mathcal{P}}(I) = \{\alpha \mid \alpha \leftarrow B_1, ..., B_m, m \geq 0 \text{ is a ground instance of a clause in } \mathcal{P} \text{ and } \{B_1, ..., B_m\} \subseteq I \text{ and second-order variables in } \alpha \text{ are bound to Skolem constants}\}$$

As in Definition 3.1, logical consequences are derived as the head atoms of ground instance of clauses which body literals are true under the interpretation considered. First-order variables also are bound to elements from the Herbrand Universe. In addition to Definition 3.1, in Definition 5.1, potential second-order variables are bound to predicate symbols: second-order variables in the body are bound to existing predicate symbols but potential second-order in the

head of clauses are bound to new Skolem constants which representing new predicate symbols. While applying the $T_{\mathcal{P}}$ operator to first-order logic programs generates atoms which are part of the Herbrand base of $\mathcal{P}$, for second-order logic programs it generates atoms which may have Skolem constants as predicate symbols in which case it extends the Herbrand base of $\mathcal{P}$.

Example 5.1 (Immediate Consequence Operator of a Definite Second-Order Program)**.** *Suppose there is a chess board with a rook* $r1$ *and a white king* $k1$ *located on the same file.* $\mathcal{P}$ *is a second-order logic program containing the* postcon *meta-rule and ground unit clauses:*

$$\begin{aligned}\mathcal{P} = \{&Q(A,B) \leftarrow R(A,B), S(B);\\ &\mathit{rook}(r1) \leftarrow;\\ &\mathit{king}(k1) \leftarrow;\\ &\mathit{white}(k1) \leftarrow;\\ &\mathit{samefile}(r1,k1) \leftarrow\}\end{aligned}$$

Applying the immediate consequence operator once to the interpretation $I = \emptyset$ *results in the following set of ground atoms which are part of the Herbrand base:*

$$T_{\mathcal{P}}(I) = \{\mathit{rook}(r1), \mathit{king}(k1), \mathit{white}(k1), \mathit{samefile}(r1,k1)\}$$

Applying the immediate consequence operator a second time makes use of the postcon *meta-rule and generates the following set of atoms:*

$$T_{\mathcal{P}}(T_{\mathcal{P}}(I)) = \{\mathit{samefileking}(r1,k1), \mathit{samefilewhite}(r1,k1)\}$$

Atoms in $T_{\mathcal{P}}(T_{\mathcal{P}}(I))$ *have respectively been generated from the following ground instances of the* postcon *meta-rule:*

$$\begin{aligned}\mathit{samefileking}(r1,k1) &\leftarrow \mathit{samefile}(r1,k1), \mathit{king}(k1).\\ \mathit{samefilewhite}(r1,k1) &\leftarrow \mathit{samefile}(r1,k1), \mathit{white}(k1).\end{aligned} \tag{5.1}$$

In these ground instances of the postcon *meta-rule, body literals are elements of* $T_{\mathbb{P}}(I)$*. The predicate names '*samefileking*' and '*samefilewhite*' are Skolem constants representing new predicate symbols*[1]*. Atoms in* $T_{\mathbb{P}}(T_{\mathbb{P}}(I))$ *have a Skolem constant as predicate name thus this second application of the immediate consequence operator has extended the Herbrand base.*

5.3.2 Predicate Invention

A MIL learner takes as input a background knowledge $B = B_c \cup M$ treated as a second-order definite program and composed of first-order definite clauses B_c and second-order definite clauses M called meta-rules. We refer in the following to T_B as the immediate consequence operator associated with the second-order program $B = B_c \cup M$ provided as input to a MIL learner. Predicates are invented as follows. The T_B operator is iteratively applied starting from the empty set. For each new atom generated from a second-order clause, the resulting meta-substitution is saved, including the Skolem constant generated. This meta-substitution can later be projected onto the corresponding meta-rule to derive the definition of the invented predicate which name is this saved Skolem constant.

Example 5.1 (Continued, Predicate Invention with the Immediate Consequence Operator)**.** *After having computed* $T_{\mathbb{P}}(T_{\mathbb{P}}(I))$*, the learner saves the following meta-substitutions corresponding to the ground instances of second-order clauses in the clauses 5.1:*

$$sub(postcon, [Q/samefileking, R/samefile, S/king])$$
$$sub(postcon, [Q/samefilewhite, R/samefile, S/white])$$

These meta-substitutions can be projected back onto the postcon *meta-rule to derive the following first-order logic program* $\mathbb{Q}$ *representing the definitions of the invented predicates* samefilek-

[1]Skolem constants are automatically generated as the concatenation of the meta-rule's name and the instantiated second-order variables in the meta-rule's body which are saved in the meta-substitution. However, Skolem constant names have been shortened for conciseness and clarity in the examples.

ing/2 *and* samefilewhite/2*:*

$$\mathcal{Q} = \{samefileking(A, B) \leftarrow samefile(A, B), king(B);$$
$$samefilewhite(a, b) \leftarrow samefile(A, B), white(B)\}$$

5.3.3 Elimination of Redundant Predicates

Successive applications of the T_B operator generate a series of predicate symbols together with their definitions. The generation of invented predicates is constrained by the language bias and background knowledge. Still, the number of predicate symbols monotonically increases with the number of iterations. To avoid cluttering the background knowledge, redundant predicates are pruned at the end of each iteration. We define a notion of equivalence of predicates based on equivalence of logic programs from success sets [Maher, 1988]. First, we refer to Definition 3.2 for the definition of the success set of a first-order logic program. We define the success set of a predicate p given a first-order logic program $\mathcal{P}$ as:

Definition 5.2 (Success set of a predicate given a first-order logic program)**.** *Assume a first-order logic program $\mathcal{P}$, the success set $SS(p, \mathcal{P})$ of a predicate p is the subset of the success set of $\mathcal{P}$ restricted to atoms of p:*

$$SS(p, \mathcal{P}) = \{\alpha \in SS(\mathcal{P}) | \alpha \text{ has predicate symbol } p\}$$

We define our notion of predicate equivalence as the equality of success sets:

Definition 5.3 (Equivalence of Predicates)**.** *Two predicates p_1 and p_2 are equivalent given a first-order program $\mathcal{P}$ if they have the same success set given $\mathcal{P}$ up to renaming of the predicate symbols p_1, p_2:*

$$SS(p_1, \mathcal{P}) =_{rename(p_1,p_2)} SS(p_2, \mathcal{P})$$

Given a predicate p_1, if there exists a predicate p_2 such that p_1 and p_2 are equivalent, p_1 is said to

be redundant with respect to p_2. Intuitively, predicates covering exactly the same set of ground atoms are not discriminative in a learning process, and adding such redundant predicates in the background knowledge is helpless to build an hypothesis.

We recall that the success set of a first-order logic program $\mathcal{P}$ is equal to the least fix point of the immediate consequence operator [Lloyd, 1987]: $SS(\mathcal{P}) = T_{\mathcal{P}} \uparrow^{\omega}$. This equality provides a simple and practical way of computing success sets, as the new atoms generated at each iteration are saved together with predicate definitions. Therefore, evaluating success sets is straightforward in our bottom-up construction.

Example 5.1 (Continued, Predicate Equivalence)**.** *The invented predicates samefileking/2 and samefilewhite/2 have the following success sets given* $\mathcal{P} \cup \mathcal{Q}$*:*

$$SS(samefileking, \mathcal{P} \cup \mathcal{Q}) = \{samefileking(r1, k1)\}$$
$$SS(samefilewhite, \mathcal{P} \cup \mathcal{Q}) = \{samefilewhite(r1, k1)\}$$

Therefore, they are considered as equivalent given $\mathcal{P} \cup \mathcal{Q}$*.*

5.3.4 Algorithm

Our algorithm for bottom-up predicate construction is presented in Algorithm 5.1. Given a second-order logic program $B = B_c \cup M$, the learner considers the restriction $B_{|E}$ of the background knowledge B_c related to the training examples E. This ensures the background knowledge considered is related to the learning task at hand. We will describe our implementation of this step in Section 5.5. The learner successively computes the immediate consequence of $B_{|E}$ according to Definition 5.1. Resulting invented predicates which are not equivalent to any current predicates following Definition 5.3 are saved in the background knowledge. Their success sets are saved such that can be used in subsequent iterations. After k iterations, the top-down learner learns a consistent hypothesis while being allowed to reuse predicates invented in bottom-up iterations. In our experiment, we will use *Metagol* as top-down learner. However, our bottom-up algorithm can be paired with any other MIL system.

Algorithm 5.1 Bottom-Up Learner

Input: second-order logic program B, examples E, number of iterations k
Output: logic program H

set $H = \emptyset$ and $I = \emptyset$
set $B = B_{|E}$ the restriction of the background knowledge B related to the examples E
for $i = 1$ to k **do**
  for all new predicates p_1 in $T_B(I)$ **do**
    if $\nexists\ p_2 \in H$ such that p_2 and p_1 are equivalent **then**
      add the definition of p_1 to H
      add atoms from $T_B(I)$ with predicate symbol p_1 to I
    end if
    $I = T_B(I)$
  end for
end for
return H

5.4 Theoretical Analysis

In the following, we call m the number of meta-rules and n_{pred} the number of initial predicate symbols available to the learner. We call k the number of bottom-up iterations performed. We restrict the scope of this work to the usual MIL program class H_2^2 defined in Definition 3.8:

Assumption 5.1 (Program class H_2^2)**.** *We assume meta-rules and hypothesised programs belong to the program class H_2^2 which consists of definite Datalog programs with dyadic predicates and at most 2 atoms in the body of each clause [Muggleton* et al.*, 2015].*

5.4.1 Number of predicate symbols introduced

The number of invented predicates monotonically grows with the number of bottom-up iterations. We first provide an upper bound over the number of predicates symbols introduced as a function of m, n_{pred} and k.

Theorem 5.1 (Number of predicate symbols introduced)**.** *We call y the column vector of powers of n_{pred}: $y = \{n_{pred}^j\}_{j=0}^{\infty}$. We call e the row vector $e = \{\delta_{j,1}\}_{j=0}^{\infty}$, where $\delta_{j,k}$ is the Kronecker delta. For all $k \in \mathbb{N}^*$, the number of predicate symbols available at the iteration k is*

upper bounded by a function u_k which is polynomial in n_{pred} and m and which is defined by:

$$\forall\ k\ \in \mathbb{N}, u_k = eT^k y \text{ with } T \text{ verifying } \forall\ j, l\ \in\ \mathbb{N} : T_{j,l} = \binom{j}{l-j} m^{l-j}.$$

Proof. We prove by induction over $k \in \mathbb{N}$ that the number of predicate symbols available at the iteration k is upper bounded by a function u_k polynomial in n_{pred} and m. For $k = 0$, the initial number of predicate symbols is $u_0 = n_{pred}$ and is polynomial in n_{pred} and m. Let $k \in \mathbb{N}$. Suppose the number of predicate symbols available at the iteration k is upper bounded by u_k which is polynomial in n_{pred} and m. The number of different clause bodies that can be constructed from u_k predicate symbols and one H_2^2 meta-rule is at most u_k^2. Then, the number of different bodies that can be constructed from m distinct H_2^2 meta-rules is at most mu_k^2, this number is the number of predicates that can be constructed in this iteration. Therefore, the number of predicate symbols available at the iteration $k+1$ is upper bounded by $u_{k+1} = u_k + m.u_k^2$ which is polynomial in n_{pred} and m. Then, for all $k \in \mathbb{N}$, the number of predicate symbols is upper bounded by u_k, which is a non-linear sequence defined by the recursive formula:

$$\begin{cases} u_0 = n_{pred} \\ u_{k+1} = u_k + m.u_k^2 \end{cases}$$

The solution to this non-linear recursive sequence is, from [Rabinovich *et al.*, 1996]:

$$\forall\ k\ \in \mathbb{N}, u_k = eT^k y \text{ with } T \text{ verifying } \forall\ j, l\ \in\ \mathbb{N} : T_{j,l} = \binom{j}{l-j} m^{l-j}$$

□

We have used in Theorem 5.1 above the notation from [Rabinovich *et al.*, 1996]. The size of matrices y, T and e has been noted as infinite to represent the fact that the size of the sub-matrix considered in the computations grows as a function of k. This size however is finite for any finite k. Moreover, we emphasise that indexes run from 0.

Example 5.2 (Bound over the number of predicates introduced)**.** *We consider $k = 2$. The row*

vector e, the matrix T and the column vector y have values:

$$e = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \end{bmatrix} \text{ and } T = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & m & 0 & 0 \\ 0 & 0 & 1 & 2m & m^2 \\ 0 & 0 & 0 & 1 & 3m \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \text{ and } y = \begin{bmatrix} 1 \\ n_{pred} \\ n_{pred}^2 \\ n_{pred}^3 \\ n_{pred}^4 \end{bmatrix}$$

Then, after $k = 2$ iterations, the number of predicate symbols is upper bounded by:

$$u_2 = eT^2y = n_{pred} + 2mn_{pred}^2 + 2m^2n_{pred}^3 + m^3n_{pred}^4$$

For a given number of iterations k, Theorem 5.1 provides an upper bound on the number of predicate symbols introduced. This upper bound is polynomial in n_{pred} and m since our predicate invention method is constrained by the metarules and primitives. In practice, this number of invented predicates also is limited by the initial background knowledge B and is polynomial in the size of the background knowledge since constructed invented predicates must be supported by ground units clauses in the background knowledge. In Section 5.5, we will describe our selection process of initial ground background clauses which is used to limit the complexity.

5.4.2 Completeness

We theoretically demonstrate within this Subsection the completeness of our predicate construction method with respect to a fragment of H_2^2 verifying the following assumption:

Assumption 5.2 (Recursions and disjunctions)**.** *We assume hypotheses in the language do not contain recursions nor disjunctions.*

We define the following iterated T_B operator for $I \subseteq \mathcal{B}_\mathcal{P}$ as:

$$T_{0,B}(I) = I \text{ and } \forall i \in \mathbb{N}^*, T_{i,B}(I) = T_B(T_{i-1,B}(I)) \cup T_{i-1,B}(I)$$

Algorithm 5.1 run for k iterations derives a program H containing predicate definitions, each having at most k non-recursive clauses. For all clauses in H, there exists a ground substitution θ of the head belonging to $T_{k,B}(\emptyset)$. In other words, H satisfies the equation below for k:

$$\forall Head \leftarrow Body \in H, \exists \theta : \begin{cases} Head\ \theta = A \\ A \in T_{k,B}(\emptyset) \end{cases} \tag{5.2}$$

Theorem 5.2 (Completeness)**.** *Assume $k \in \mathbb{N}$ and a theory H within the hypothesis space defined by the primitives and meta-rules provided. H satisfies Equation 5.2 with parameter k only if it is derivable in k iterations of Algorithm 5.1.*

Proof. We prove by induction over $k \in \mathbb{N}$ that H satisfies Equation 5.2 with parameter k only if it is derivable in k iterations of Algorithm 5.1. For $k = 0$, $H = \emptyset$ and our statement trivially holds. Let $k \in \mathbb{N}$. We assume theories satisfying Equation 5.2 with parameter k are derivable in k iterations of Algorithm 5.1. We consider a theory H satisfying Equation 5.2 with parameter $k+1$. For all $Head \leftarrow Body \in H$, there exists a ground substitution θ such that $Head\ \theta = A$ and $A \in T_{k+1,B}(\emptyset)$. All literals from $Body\ \theta$ are elements of $T_{k,B}(\emptyset)$ by definition of $T_{k+1,B}$ and T_B. Then the set of predicate definitions associates with literals from $Body\ \theta$ satisfy Equation 5.2 thus is derivable in k iterations of Algorithm 5.1. Performing one more iteration of Algorithm 5.1 derives $A = Head\theta$ and the clause $Head \leftarrow Body$ is saved in H. Then, clauses from H are derivable in $k+1$ iterations of Algorithm 5.1. □

Theorem 5.2 shows our method is complete with respect to a non-recursive fragment of H_2^2. In other words, our bottom-up construction method can construct all k clauses definitions supported by the background knowledge in k bottom-up iterations. If the top-down learner chosen also is complete, then the system combining the bottom-up and top-down learner is complete.

5.4.3 Sample complexity

The construction of substrate predicates helps to reduce the length of surface hypotheses, which can reduce the learning complexity. We show that the construction of substrate predicates can in turn reduce the sample complexity.

Proposition 5.1 (Sample Complexity gain [Cropper, 2019])**.** *We assume the target hypothesis expressed in its minimal form has n clauses with standard MIL. Let l_k be the reduction over the number of clauses in the minimal representation of the target hypothesis after k bottom up iterations, that is the minimal number of clauses required to express the target hypothesis after k bottom-up iterations is $n - l_k$. We call $n_{pred,k}$ the number of predicate symbols available after k bottom-up iterations. We assume $n_{pred,0}$ initial predicates, m meta-rules in H_2^2, an error level $\epsilon > 0$ and a confidence level $1 - \delta$ with $\delta > 0$. Then, the number of examples required to achieve an error at most ϵ with confidence at least $1 - \delta$ is reduced after k bottom-up iterations when:*

$$n \ln(n_{pred,0}) > (n - l_k) \ln(n_{pred,k})$$

Our bottom-up method revises the hypothesis space. The introduction of new predicate widens the hypothesis space but also reduces the length of surface hypotheses which reduces the search depth. Proposition 5.1 states that the sample complexity is reduced when the search depth reduction offsets the cost of adding new predicates. It is worth noting that, by completeness from Theorem 5.2, it is guaranteed that the number of clauses required to express a target theory is reduced by at least $l_k = k$ after k bottom-up iterations when the target theory expressed in its minimal form has $n > k$ clauses with standard MIL. Indeed, all k clauses definitions are constructible in k iterations. After they are made available to the top-down learner, it remains to learn $n - k$ clauses. Therefore, reduction over the search depth is ensured.

5.5 Implementation

5.5.1 Sampling of background knowledge facts

Our bottom-up predicate invention method has a complexity polynomial in the number of initial predicate symbols n_{pred} and in the number of meta-rules m according to Theorem 5.1. The number of invented predicates also is restricted by the background knowledge B available since invented definitions must be supported by facts from the background knowledge. Therefore, in order to limit the complexity and in order to generate predicates relevant to the learning task, Algorithm 5.1 considers the background knowledge $B_{|E}$ which is the restriction of the background knowledge B related to the examples E. This allows to constrain the search for invented predicates. $B_{|E}$ is a set of ground unit clauses sampled from the examples and initial predicates as follows. First, the ground terms in the examples are extracted. Next, initial primitive predicates are iteratively applied to these ground terms used as input variable. Each successful resolution is saved as a background fact. If a resolution generates an output ground term, this output ground term is saved and can be reused in further resolutions as input to build new facts. This process is repeated. The number of iterations is set to a function of the number of bottom-up iterations performed. Examples can be sampled uniformly at random to ensure the size of the initial background knowledge is not too large. This process generates a set of connected background facts which are relevant to the training examples. This set of background fact is saved as $B_{|E}$.

5.5.2 Applying the T_B operator

New facts and invented predicate definitions are generated by applying the T_B operator from Definition 5.1 to $B_{|E}$. Bodies of meta-rules in M are resolved against the background knowledge. In order to limit redundancy of these resolution proofs, we ensure that each new proof reuses at least one fact proved in the last iteration. Hence, a proof executed at the bottom-up iteration i will not be executed again at iteration j, with $j > i$.

5.5.3 Generation of unique Skolem constants

The successful resolution of the body of a second-order clause produces a head atom for which second-order variables are bound to new Skolem constants. Skolem constants are built by concatenating the meta-rule's name and the instantiated second-order variables in the meta-rule's body. This process ensures the uniqueness of Skolem constants. If two predicates are equivalent with respect to Definition 5.3, the predicate with the shortest number of concatenated names is saved since it has a simpler definition by construction.

5.6 Experiments

5.6.1 Experimental Hypotheses

We experimentally test within this section whether the use of bottom-up iterations can improve learning performance[2]. We evaluate the learning performance as the predictive accuracy, the sample complexity and the learning time. Therefore, we investigate the following Experimental Hypotheses:

Experimental Hypothesis 5.1. *Augmenting MIL systems with bottom-up iterations can improve predictive accuracies compared to standard MIL.*

Experimental Hypothesis 5.2. *Augmenting MIL systems with bottom-up iterations can reduce the sample complexity compared to standard MIL.*

Experimental Hypothesis 5.3. *Augmenting MIL systems with bottom-up iterations can reduce learning times compared to standard MIL.*

To test these Experimental Hypotheses, we use the state-of-the-art MIL learner *Metagol*. We compare the regular version of *Metagol* versus *Metagol* augmented with bottom-up iterations. In each experiment, we provide both learning systems with the same background knowledge

[2]The code for reproducing the experiments is available at `https://github.com/celinehocquette/bottom_up.git`

containing the same primitives and meta-rules. Therefore, the only variable is the learning system. We associate to the previous Experimental Hypotheses the following Null Hypotheses that we will test:

Null Hypothesis 5.1. *Augmenting Metagol with bottom-up iterations can not improve predictive accuracies compared to standard Metagol.*

Null Hypothesis 5.2. *Augmenting Metagol with bottom-up iterations can not reduce the sample complexity compared to standard Metagol.*

Null Hypothesis 5.3. *Augmenting Metagol systems with bottom-up iterations can not reduce learning times compared to standard Metagol.*

5.6.2 Rook protected in Chess Endgame KRK

KRK denotes the chess ending with white having a king and a rook and black having a king. An optimal strategy is known and its correctness has been demonstrated [Bratko, 1978]. This optimal strategy involves reducing the area available to the black king using the white rook. An essential feature of this strategy is to ensure the safety of the white rook at all times. The protection of this white rook can be achieved using the white king. This experiment considers the task of learning whether the white rook is protected by its king. An example of a situation in which the white king protects its rook is represented in Figure 5.1a.

Material The state of the board is a list of non-empty cells. Cells are atoms of the form *c(X,Y,Colour,Type)* and encode the current position with rank X and file Y of the piece of colour *Colour* (white or black) and type *Type* (rook or king). We provide both learners with 6 primitives: *piece/2* which extracts non-empty cells from states, *rook/1* and *king/1* which hold for non-empty cells with a piece of type rook or king respectively, *white/1* and *black/1* which hold for non-empty cells with a piece of color white or black respectively, *distance1/2* which holds when the arguments are two pieces separated by a Chebyshev distance of 1. We provide both learners with four meta-rules shown in Table 5.1. These meta-rules belong to H_2^2. The

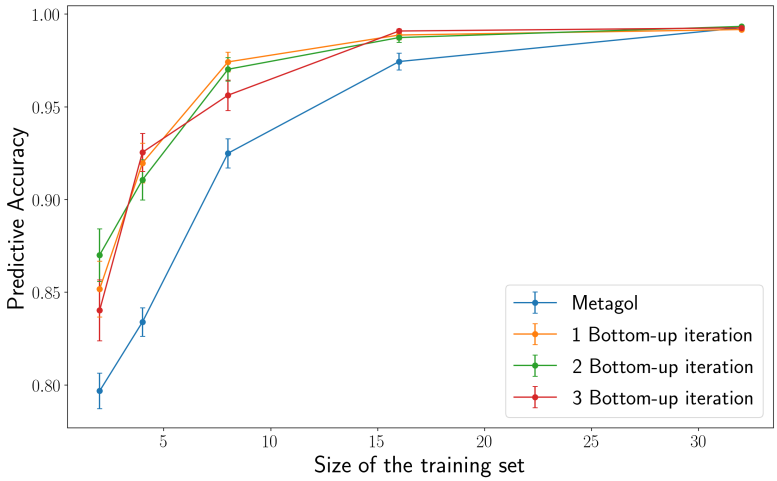
Experiment	Name	Meta-rule
1	*postcon*	$Q(a,b) \leftarrow R(a,b), S(b).$
1	*conj*	$Q(a) \leftarrow R(a), S(a).$
1	*conj2*	$Q(a) \leftarrow R(a,b), S(a,b).$
1 and 2	*chain*	$Q(a,b) \leftarrow R(a,c), S(c,b).$

Table 5.1: Meta-rules used in the experiments: the letters Q, R, S denote existentially quantified second-order variables and the letters a, b, c universally quantified first-order variables.

target theory is shown in Figure 5.1b. It states that the rook is protected by its king in a board S if there is a white king and a rook in S and if these two pieces are at distance 1 of each other.

Methods Training instances are instances of the form rook_protected(S) where S is a board state. Positive examples are generated by placing the white rook and the black king on different squares randomly selected on an empty board. The white king is placed on a random empty square at distance 1 from the rook. Negative examples are generated by altering an attribute (rank, file, colour or type) selected at random of either the white king or the rook in a positive example such that the target theory does not hold for the resulting state. We perform between 1 and 3 bottom-up iterations. Initial facts are built from a sample of size 1 of the positive examples. We measure the number of correct classifications over a set of 300 test instances generated following the same process as the training set. Training and testing sets are built with half positive, half negative examples therefore the default accuracy is 0.5. We compare accuracies and learning times versus the number of training examples. We measure the standard error of the mean over 100 repetitions.

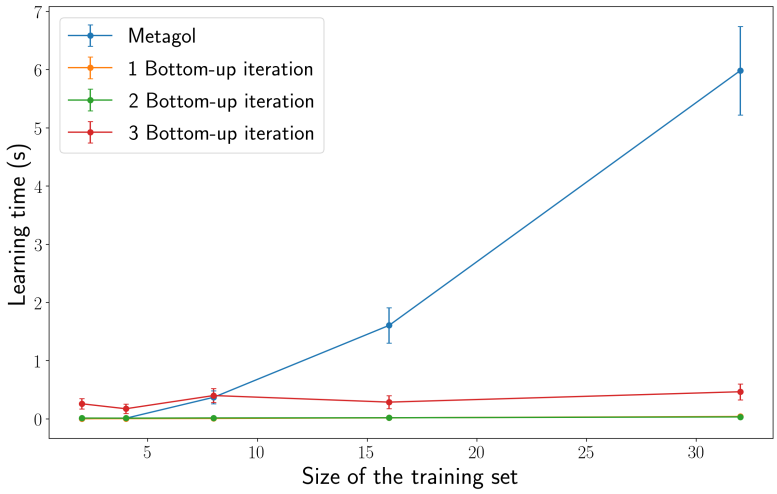
Results Accuracy results are presented in Figure 5.2a. These results show that the regular version of *Metagol* has worst predicative accuracy and converges slower compared to *Metagol* augmented with any number of bottom-up iterations tested. A T-test suggests that the difference in accuracy is statistically significant ($p < 0.05$) for a training set with up to 16 instances thus refuting Null Hypothesis 5.1. Sample complexity results are detailed in Figure 5.2b: these results show that performing bottom-up iterations can reduce the sample complexity: the reg-



(a) KRK: accuracy

Accuracy	*Metagol*	k=1	k=2	k=3
0.85	5	2	2	3
0.9	7	4	4	4
0.95	13	7	7	8
0.98	21	13	13	14

(b) Experimental Sample Complexity vs the number of bottom-up iterations k



(c) KRK: learning time

Figure 5.2: KRK experiment results: Learning "rook protected by the king"

ular version of *Metagol* requires at least 50% more training examples than *Metagol* augmented with bottom-up iterations. We thus refute Null Hypothesis 5.2. The surface hypothesis from Figure 5.1b has a size typically reduced from 5 to 3 clauses after $k = 1$ bottom-up iteration and to 1 clause for $k = 2$ and $k = 3$. It is better than the expected reduction of 1, 2 and 3 clauses for $k = 1$, $k = 2$ and $k = 3$ respectively guaranteed by completeness from Theorem 5.2. The number of predicates available after the iterations $k = 1$, $k = 2$ and $k = 3$ typically is at most $p_1 = 18$, $p_2 = 42$ and $p_3 = 71$ respectively. These experimental values are consistent with Proposition 5.1 which theoretically shows than bottom-up iterations, owing to the generation of reusable predicates, can reduce the sample complexity. One can observe that these numbers of invented predicates are much lower than the worst case scenario bound provided by Theorem 5.1 which guaranteed $p_1 \leq 150$, $p_2 \leq 90150$ and $p_3 \leq 3.3 \times 10^{10}$. This difference can be explained by the requirement that invented predicates must be supported by facts in the background knowledge and must be consistent with the language bias which restricts the number of predicates invented in practice. Finally, learning times are presented in Figure 5.2c. These results show that *Metagol* augmented with bottom-up steps requires significantly shorter learning times compared to the regular version of *Metagol*. We thus refute Null Hypothesis 5.3. However, one can observe than learning times are greater for 3 iterations compared to 1 and 2 iterations. This can be explained by the fact than more predicates have been generated and the search is unnecessarily more expensive. Performing more than 2 bottom-up iterations ($k > 2$) is not beneficial for this experiment as it can not further reduce the size of the target theory but increases learning times.

5.6.3 String transformations

Materials We consider 94 string transformation problems evaluated in [Cropper, 2019] and inspired from real-world spreadsheet problems [Gulwani, 2011; Lin *et al.*, 2014]. This dataset contains 10 positive examples for each problem. Each example is an atom of the form $task(s1, s2)$ where $task$ is the problem name and $s1$ and $s2$ are input and output strings respectively. Two examples of tasks are shown in Tables 5.2a and 5.2b. For each run and each task, one example

Input	Output
James Brown	BROWN
David Batty	BATTY

(a) Task p9: examples

Input	Output
James Brown	JB
Joanie Faas	JF

(b) Task p39: examples

```
p39(A,B):-copy1skipwordskip1copy1(A,C),skiprest(C,B).
copy1skipwordskip1copy1(A,B) :- copy1skipword(A,C), skip1copy1(C,B).
copy1skipword(A,B):-copy1(A,C), skipalphanum(C,B).
skip1copy1(A,B):- skip1(A,C), copy1(C,B).
```

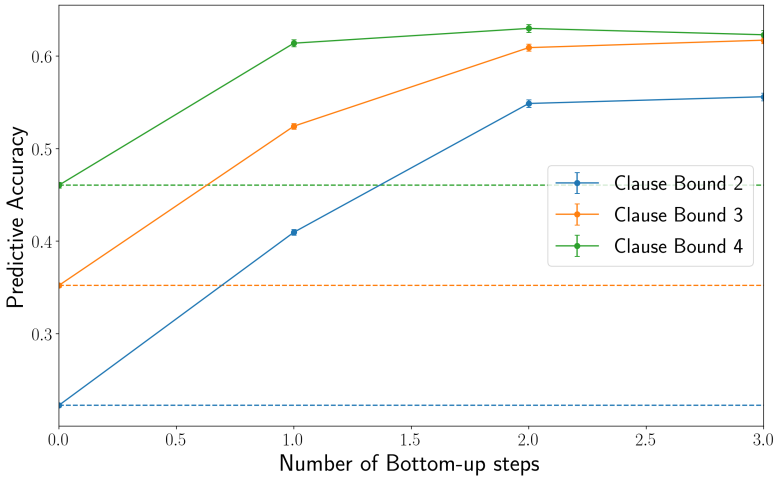
(c) Hypothesis learned for task p39: initial predicates are represented in bold. Predicates have been renamed for clarity.

Table 5.2: String Transformation Experiment

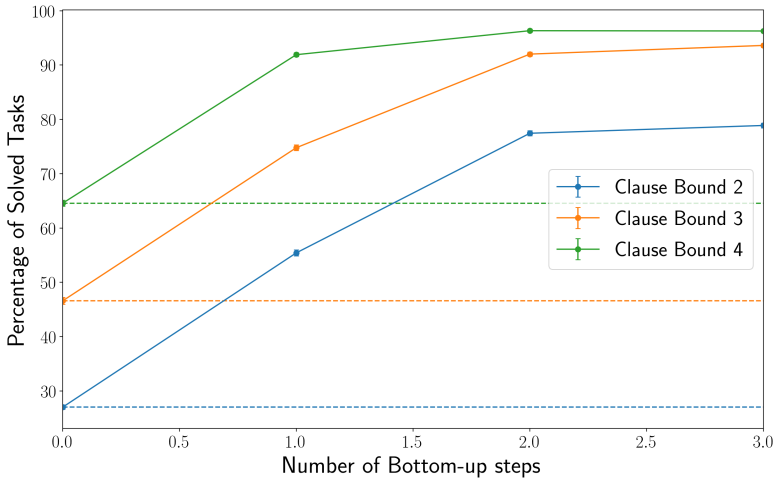
is randomly selected to form the training set, the remaining nine being left for testing. We provide both systems with the same background knowledge from [Lin *et al.*, 2014] containing the nine primitive predicates *copyalphanum/2*, *copy1/2*, *write_point/2*, *skipalphanum/2*, *skip1/2*, *skiprest/2*, *make_lowercase/2*, *make_uppercase/2*, *make_uppercase1/2* and one meta-rule which is the *chain* meta-rule represented in Figure 5.1. The language is a subset of H^2_2.

Methods We set the size of the top-down learner search space to $n \in [2, 4]$ clauses. For $k \in [0, 3]$, k bottom-up iterations are performed. A time-out is set to 10 minutes. If a learning task fails and no hypothesis is returned, the accuracy is set to 0. We measure the standard error of the mean over 20 repetitions for the 94 tasks. As in [Lin *et al.*, 2014; Cropper, 2019], a functional restriction is set to compensate for the lack of negative examples.

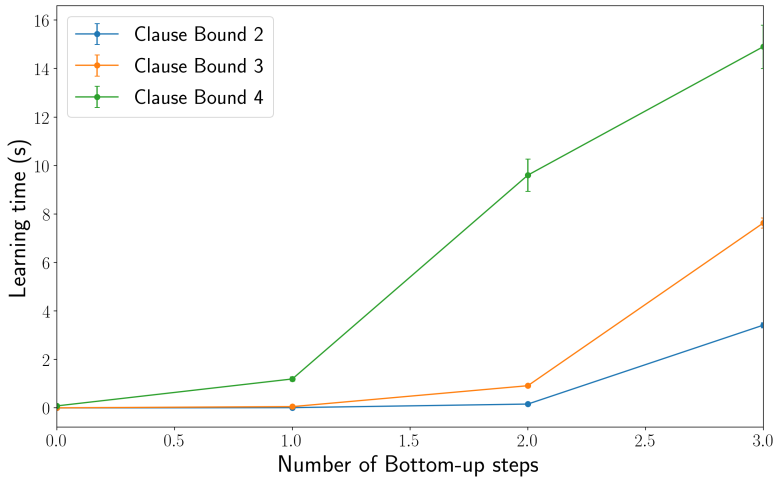
Results Results are presented in Figure 5.3. Figure 5.3a shows that the accuracy increases with the number of bottom-up iterations for any clause bound tested, outperforming the regular version of *Metagol* represented with the dotted horizontal lines. Thus, we confirm the refutation of Null Hypothesis 5.1. Figure 5.3b represents the percentage of tasks solved versus the number of iterations. Both the accuracy and the percentage of tasks solved for a clause bound of n and after k bottom-up iterations are greater than that's of *Metagol* for a clause bound of $n + k$. Indeed, k bottom-up iterations can in practice reduce the target theory by more than k clauses which is more than the guarantee provided by Theorem 5.2. Learned hypotheses generally are



(a) Accuracy



(b) Percentage of tasks solved



(c) Learning time of solved tasks

Figure 5.3: String transformation experiment results: baselines (dotted lines) correspond to the regular version of *Metagol*, results for *Metagol* augmented with bottom-up steps are represented with solid lines

characterised by a high usage of predicates invented in bottom-up iterations. For example, the hypothesis learned for the task 39 is shown in Table 5.2c. Respectively $k = 1$ or $k = 2$ bottom-up iterations are enough to build the bottom 2 or 3 invented substrate predicates in which case the surface hypothesis has respectively only 2 or 1 clauses. Conversely, the regular version of *Metagol* requires 4 clauses to learn the same hypothesis. The reduction of the number of clauses for this example is in practice larger than the worst-case theoretical result: completeness from Theorem 5.2 guaranteed a reduction of at least 1 clause for $k = 1$ and 2 clauses for $k = 2$. The number of predicates available is at most $p_1 = 38$, $p_2 = 143$ and $p_3 = 166$ for $k = 1$, 2 and 3 respectively. These numbers are much lower than the bounds provided by Theorem 5.1 which guaranteed $p_1 \leq 90$, $p_2 \leq 8190$ and $p_3 \leq 6.8 \times 7$. Finally, Figure 5.3c represents learning times. Learning times increase with the number of bottom-up iterations, especially for larger clause bounds. Augmenting the number of iterations makes available more background knowledge up to the point it is unnecessary and starts to hinder the search.

5.6.4 Discussion

We have demonstrated over two domains our method can improve learning performance. Our method increases the breadth of the search, through the addition of more predicate symbols, but decreases the depth of the search, as hypotheses are shorter thanks to the reuse of predicates. However, we have observed in Figure 5.3c that learning times can increase with the number of iterations. This result highlights a limitation of our approach: the background knowledge is monotonically increasing with the number of iterations, up to the point the breadth of the search becomes too large, which hinders the search. This prevents scalability for large number of iterations with large initial background knowledge. Future work is needed to identify predicates from lower-levels which can be ignored when higher level predicates are made available.

5.7 Future Work

We identify the following limitations of these contributions which could be addressed as future work.

Learning more complex theories We have restricted the scope of this work with Assumption 5.2 to non-recursive definitions containing no disjunctions. Future work needs to be conducted to extend this work to learn disjunctive definitions in bottom-up iterations. Learning disjunctions could then allow learning recursions. Recursive definitions need a base case. The base case can be given in the background knowledge or could have been invented in a previous bottom-up iteration. A base case being available, a recursive clause can be constructed using the *tailrec* meta-rule in the case that the body literals can be proved against the current background knowledge. There are no unbound second-order variables in the head of the *tailrec* meta-rule therefore no new Skolem constant need to be generated. However, multiple such recursive clauses for the same predicate could be generated, and relevant subsets of these clauses must be selected as appropriate disjunctions. We suggest the use of higher-order definitions [Cropper *et al.*, 2020b] could allow learning more complex definitions involving disjunctions or recursions. Higher-order definitions can abstract away the complexity and make the recursion implicit in the higher-order definition. Alternatively, meta-rules could be extended into meta-programs. While meta-rules are single second-order clauses, a meta-programs represent multi-clause templates. The use of meta-programs has been investigated to learn divide-and-conquer programs [Flener and Yılmaz, 1999] and could allow the representation of more complex programs in bottom-up iterations.

Theoretical Analysis Proposition 5.1 specifies a condition under which performing bottom-up iterations reduces the sample complexity. However, this proposition does not predict a priori when this condition will be fulfilled. For instance, we have experimentally observed in the KRK experiment that performance does not continue to improve after $k = 2$ iterations. In addition, performing too many iterations can overcrowd the background knowledge and hinder the search.

For instance, learning time increases with the number of iterations in the string transformation experiment. Future work is needed to theoretically predict the optimal number of bottom-up iterations given a search space. This characterisation could be based on the density of relevant predicates in the search space and on a bound over the number of invented predicates tighter than the one provided by Theorem 5.1.

Moreover, preliminary experiments suggest our method is more effective for target theories for which the calling diagram has a balanced tree structure instead of a more complex directed cyclic or acyclic graph. The former benefit from a larger reduction over the size of the surface hypothesis after bottom-up iterations than the latter. Future work is needed to theoretically characterise target theories for which this bottom-up method is more effective. This characterisation will be based on the degree of reuse of invented predicates in the calling diagram.

Sampling of initial facts Initial facts are sampled from the constants provided in the examples and from the primitive symbols. However, the complexity depends on the number of these initial facts and thus depends on the number of constant symbols which prevents scalability. Future work is needed to investigate a more efficient selection of relevant initial facts. We suggest these facts could be identified with relational path-finding [Richards and Mooney, 1992; Ong *et al.*, 2005]. Considering the background knowledge forms an undirected graph, relational path-finding finds paths by successive expansion around the nodes associated with the constants in the positive examples. Another suggestion is to completely eliminate the complexity dependence over the number of constant symbols by lifting initial grounded terms to first-order terms which generalises this set of grounded terms.

More efficient representations We have implemented computations as Prolog proofs. Future work could develop more efficient computations approaches to scale up to larger domains. For instance, more efficient logical reasoning could be realised with linear algebraic multiplications of programs and interpretations represented in a vector space [Sakama *et al.*, 2017; Nguyen *et al.*, 2021]. Alternative data-structure could also be investigated. In [Gulwani, 2011], a directed acyclic graph is used to represent and manipulate traces representing different ways

of generating an output string from the input string. Similarly, we suggest representing knowledge with a Directed Acyclic Word Graph [Blumer *et al.*, 1985] built from the training data and the background knowledge. A main advantage of this structure is its construction and information retrieval efficiency: a Directed Acyclic Word Graph can be built in time linear in the size of the input string.

5.8 Summary

We have introduced in this Chapter a new method for revising the hypothesis space in MIL via automated predicate invention. This method performs extensive predicate invention in a bottom-up fashion. It thus generates a set of reusable predicates representing intermediate concepts hierarchically composable. Predicates are invented as generalisation of the background knowledge with an extension of the immediate consequence operator for second-order logic programs. Predicate invention relies on search constrained by the meta-rules, primitives and background knowledge. A new predicate is generated only if it is supported by facts from the background knowledge which are related to the examples. We have introduced a predicate selection method based on equivalence of success sets to prune redundant predicates. We have theoretically derived an upper bound over the number of predicates invented. We have theoretically demonstrated our bottom-up method is complete with respect to a fragment of dyadic Datalog. Moreover, we have theoretically and experimentally demonstrated that this method reduces the number of clauses to be learned by the top-down learner, which can reduce the sample complexity. Our experimental results have demonstrated that our method can improve predictive accuracies and reduce the sample complexity and learning times.

This Chapter has introduced a method to revise the hypothesis space and has demonstrated this method can reduce the sample and learning complexity in MIL, thus supporting Subthesis S.2. The next Chapter 6 extends these contributions to methods that revise both the instance space and the hypothesis space thus investigates Subthesis S.3. While this Chapter has focused on learning relational features in games, Chapter 6 will focus on learning optimal game strategies.

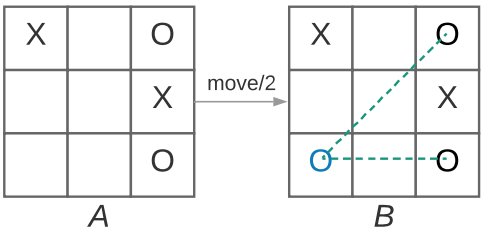
Chapter 6

Meta-Interpretive Learning of Game Strategies

In this Chapter, we investigate Subthesis S.3 and introduce a method for revising both the instance space and the hypothesis space to reduce the sample complexity and improve the learning performance in MIL and for learning game strategies. This Chapter is based on the works published in [Muggleton and Hocquette, 2019; Ai *et al.*, 2021]. Sections 6.1 to 6.5.3 are based on [Muggleton and Hocquette, 2019] and Sections 6.5.4 to 6.5.5 on [Ai *et al.*, 2021].

6.1 Introduction

Deep reinforcement learning systems have been demonstrated capable of mastering two-player games such as Go [Silver *et al.*, 2016; Silver *et al.*, 2017], Chess [Silver *et al.*, 2018], Shogi [Silver *et al.*, 2018], StarCraft [Vinyals *et al.*, 2019]outperforming world-class human players. However, these systems 1) generally require a very large training set to converge toward a good strategy, 2) do not provide transferability of the learned strategies to other games and 3) are not easily interpretable as they provide limited explanation about how decisions are made [Garnelo *et al.*, 2016; Marcus, 2018].



(a) Board example of a double attack, this is an optimal move for O

```
win_2(A,B):-win_2_1_1(A,B),not(win_2_1_1(B,C)).
win_2_1_1(A,B):-move(A,B),not(win_1(B,C)).
win_1(A,B):- move(A,B),won(B).
```

(b) Logic program describing an optimal strategy

Figure 6.1: Noughts-and-Crosses: example of optimal move for O from state A to state B. For all moves of X from state B, O can win in one move. This statement can be expressed with the logic program presented: O makes a move such that X cannot immediately win nor make a move that blocks O, see Figure 1.3.

We demonstrate in this Chapter how machine learning strategies as logic programs with MIL can overcome these limitations. For example, an optimal strategy for playing Noughts-and-Crosses is to create double attacks when possible. We refer to the example of double attack introduced in Section 1.1.5 in Chapter 1 and presented again in Figure 6.1a. Player O makes a move from board A to board B which poses two threats for the player X. These threats are represented in green. This move results in a forced win and is an optimal move for O. The rules presented in Figure 6.1b represent this strategy. A, B and C are variables which represent state descriptions and encode the board together with the current player. These rules state that a move by the current player from state A to state B is a winning move if the opponent cannot immediately win and if the opponent cannot make a move to prevent an immediate win by the current player. The rules provide an understandable strategy for winning in two moves. Moreover, these rules are transferable to more complex games as they are generally true for describing the concept of multiple simultaneous attacks. Finally, each of these rules is applicable to a range of board states and thus this strategy is a generalisation over states. In this sense, this strategy is represented compactly, therefore it is easier to learn and in turn requires a smaller sample complexity.

We introduce in this Chapter a new logical system called *MIGO* (*Meta-Interpretive Game*

Ordinator)[1] designed for learning two-player game optimal strategies of the form presented in Figure 6.1b. *MIGO* benefits from a strong inductive bias which provides the capability to learn efficiently from a few examples of games played only. Moreover, learned strategies are generally true for all two-player games which provides straightforward transferability to more complex games. Finally, learned hypotheses are provided in symbolic form which allows their interpretation.

MIGO is a MIL learner. MIL supports predicate invention which makes it suitable for learning game strategies. *MIGO* additionally supports Dependent Learning [Lin *et al.*, 2014]. The learning operates in a staged fashion: simple definitions are first learned and added to the background knowledge allowing them to be reused during further learning tasks. Thus, increasingly complex definitions are hierarchically constructed. For instance, *MIGO* first learns a simple definition of *win_1/2* for winning in one move. Next, a predicate *win_2/2* describing the concept of winning in two moves can be built from *win_1/2* as shown in Figure 6.1b.

Owing to tractability considerations Minimax Regret of a learning system cannot be evaluated in complex games. Therefore, in this Chapter we consider simple evaluable games (Noughts-and-Crosses and Hexapawn) in which Minimax Regret can be efficiently measured and be used as a measure for evaluating learning performance. We use these evaluable games to compare Cumulative Minimax Regret for variants of both standard and deep reinforcement learning against two variants of *MIGO*. Our results demonstrate that substantially lower Cumulative Minimax Regret can be achieved by *MIGO* compared to the variants of reinforcement learning tested. Additionally, rules learned by *MIGO* are demonstrated to achieve significant transfer learning in both directions between Noughts-and-Crosses and Hexapawn. Finally, *MIGO*'s learned rules are shown to be relatively easy to comprehend and to provide some comprehensibility.

Specifically, the contributions of this Chapter are as follows. We introduce a new system called *MIGO* for learning optimal two-player-game strategies (Section 6.3). We provide and describe its implementation (Section 6.4). We experimentally demonstrate *MIGO* achieves significantly smaller Cumulative Minimax Regret than reinforcement learning systems, that

[1] From the children's game-playing phrase *My go!* and the literal translation into English of the French word *Ordinateur* which means computer.

learned strategies are transferable to more complex games and that learned rules provide some form of comprehensibility (Section 6.5).

6.2 Related Work

6.2.1 Learning Game Strategies

Various early approaches to learning a game strategy [Shapiro and Niblett, 1982; Quinlan, 1983] used the decision tree learner ID3 to classify minimax depth-of-win for positions in chess end games. These approaches used a set of carefully selected board attributes as features. Conversely, we investigate learning game strategies from a set of three relational primitives (*move/2*, *won/1*, *drawn/1*) representing the minimal information a human would expect to know before playing a two-person game. Moreover, it has been reported that chess experts had difficulty understanding decision trees learned [Quinlan, 1983] due to their high complexity which made them opaque to humans [Michie, 1983]. Methods for simplifying decision trees without compromising their accuracy have been investigated [Quinlan, 1987] on the basis that simpler models are more comprehensible to humans. A later ILP approach learned optimal chess endgame strategies at depth 0 or 1 [Bain and Muggleton, 1994]. An informal complexity constraint limiting the number of clauses used in any predicate definition to 7 ± 2 clauses was applied. This number is based on Miller's [Miller, 1956] experimental demonstration of a limit of around 7 chunks in human short-term memory. Also, Aleph [Srinivasan, 2001] and TILDE [Blockeel and De Raedt, 1998] have been demonstrated to outperform a SVM learner on the Bridge opening bid problem [Legras *et al.*, 2018]. However, examples in the aforementioned works are board positions taken from a minimax database [Shapiro and Niblett, 1982; Quinlan, 1983; Bain and Muggleton, 1994] or labelled by human experts [Legras *et al.*, 2018]. Conversely, *MIGO* learns from game play. Moreover, transfer learning has not been examined for these related works.

As described in Subsection 2.1.3, reinforcement learning has been widely used for learning game

strategies. Reinforcement learning systems typically involve learning Q-values representing the quality of each possible action from each state. Relational reinforcement learning [Džeroski *et al.*, 2001; Tadepalli *et al.*, 2004] combines reinforcement learning with relational representations. They also learn a policy as a Q-function. Conversely, the learning framework *MIGO* is not based upon the identification of Q-values but aims at deriving hypotheses as logic programs describing an optimal strategy. Also, by opposition with our approach, relational reinforcement learning approaches do not involve predicate invention to extend the vocabulary provided to the user and automatically discover new concepts. Finally, while most relational reinforcement learning systems aim at learning single agent policy [Driessens and Džeroski, 2004; Sarjant *et al.*, 2011], *MIGO* is designed to learn two-player games in adversarial environment.

6.2.2 Transfer Learning

The idea of transfer learning is that experience gained while learning to perform one task can help improve learning performance in a related but different task [Taylor and Stone, 2009]. Rule Transfer [Taylor and Stone, 2007] is a transfer algorithm that first learns with the propositional learner RIPPER [Cohen, 1995] rules summarizing a policy. Rules are then transformed via hand-coded inter-task mappings so that they can apply to qualitatively different tasks. Conversely, *MIGO* learns first-order rules which are applicable to a wide range of states and thus are easily transferable to similar domains. First-order rules identifying useful skills [Torrey *et al.*, 2006] or representing strategies [Torrey *et al.*, 2007] are learned with the ILP system Aleph [Srinivasan, 2001]. Skills are conditions under which an agent should take an action while strategies are action plans composing skills together. Rules can be leveraged as advices which are incorporated inside the Q-function [Torrey *et al.*, 2006]. However, rules first need to be propositionalised. Alternatively, rules can be leveraged as demonstration [Torrey *et al.*, 2007]: the learner begins by executing the previous strategy for a set of episodes instead of exploring randomly, during these episodes it updates the Q-values with the score of the rules used. Conversely, *MIGO* transfers first-order rules representing strategies and transfer learning within *MIGO* is not based on the update or use of Q-values. Transfer Learning of heuristic pat-

tern concepts has been demonstrated between simple games such as Tic-tac-toe, Connect4 and Connect5 [Sato *et al.*, 2015]. Unlike *MIGO*, this approach does not learn to play from scratch, but, instead, uses an alpha-beta player with a heuristic function based on set of specialised concepts learned using the ILP system Aleph [Srinivasan, 2001].

6.3 Theoretical Framework

6.3.1 Credit Assignment

One can evaluate the success of a game by looking at its outcome expressed as a reward. The Credit Assignment Problem is the problem of redistributing this final reward to the various moves performed before reaching this outcome. As described in Subsection 2.1.3, reinforcement learning systems usually tackle this Credit Assignment Problem by adjusting parameter values associated with the moves responsible for the reward observed. Conversely, we introduce theorems for identifying moves that necessarily are positive examples for the task of winning and drawing. A game G is represented as a sequence of board states $G = \{B_0, B_1, B_2...\}$. A game may start from any legal state B_0. Let G be a game played between two players, the learner P_1 and its opponent P_2. We make the following assumption regarding the opponent:

Assumption 6.1 (Optimal Opponent)**.** *The learner P_1 plays against an opponent P_2 that follows an optimal strategy.*

We consider the following natural ordering over the different possible outcomes for P_1:

$$won \succ drawn \succ lost$$

The following Lemma states the expected outcome of P_1 is monotonically decreasing:

Lemma 6.1 (Decreasing Expected Outcome)**.** *The expected outcome of P_1 cannot increase during a game G.*

Proof. P_2 plays optimally. Therefore any move of P_2 maintains or lowers the expected outcome of P_1 throughout G. Therefore P_1 cannot increase its outcome. □

Lemma 4.1 states that the expected outcome of the learner can only decrease or be maintained throughout a game. We now demonstrate the theorems below to identify positive examples for the tasks of winning and drawing given Assumption 6.1 and Lemma 4.1:

Theorem 6.1 (Winning Positive Example)**.** *If the outcome of a game G is won for P_1, then every move of P_1 in G is a positive example for the task of winning.*

Proof. Let G be a winning game sequence . We suppose there exists within G a move of P_1 from a board state B_i to a board state B_{i+1}, with $i \geq 0$, such that this move is a negative example for the task of winning. Then the expected outcome of B_i is won and the expected outcome of B_{i+1} is strictly lower with respect to the ordering $\succ$. Following Lemma 4.1, the outcome of the game is strictly lower than won, which leads to contradiction with the outcome observed. □

Theorem 6.2 (Drawing Positive Example)**.** *We additionally assume an accurate strategy S_W for winning has been learned by the learner P_1. If the outcome of the game $G = \{B_0, B_1, ...\}$ is drawn and if the execution of S_W on the initial board state B_0 fails, then any move played by either P_1 or P_2 in G is a positive example for the task of drawing.*

Proof. Let G be a drawing game sequence starting from B_0. The initial position B_0 does not have an expected outcome of won for P_2 otherwise the outcome would be won for P_2 since it plays optimally according to Assumption 6.1. The initial position B_0 neither has an expected outcome of won for P_1 otherwise the execution of S_W on B_0 would not fail. Therefore, the expected outcome of B_0 necessarily is drawn. Since the outcome also is drawn, it follows from Lemma 4.1 that every position reached during the game has an expected outcome of drawn and that every move of both players is a positive example for the task of drawing. □

We recall strategies, such as S_W, are represented as logic programs. Then, the execution of the strategy S_W on the initial board state B_0 fails means that there are no moves consistent with

the strategy S_W when providing B_0 as input.

Theorem 6.1 demonstrates that when the outcome is won for P_1 and because the opponent plays optimally according to Assumption 6.1, then the expected outcome necessarily is maintained throughout the game as won for P_1. Theorem 6.2 demonstrates that when the outcome is drawn, if the initial board state does not have an expected outcome of won for P_1, then the expected outcome necessarily is maintained throughout the game as drawn. However, these theorems cannot be further generalised. For instance, an outcome of won for P_2 might be the consequence of a mistake of P_1 who does not play optimally. In this case, a prefix of the game sequence might have an expected outcome of won for P_1 or drawn and moves of P_2 in this prefix do not have an expected outcome of won for P_2 thus are not positive examples for the task of winning. Similarly, an outcome of drawn might be the consequence of a mistake of P_1 if the initial board state has an expected outcome of won for P_1, in which case a prefix of the game sequence which length is unknown does not contain positive examples for the task of drawing.

We highlight that Theorems 6.1 and 6.2 identify positive examples but do not provide any negative examples for *win/2* or *draw/2*. These theorems do not locate moves for which the expected outcome decreases. Therefore, the learning system we consider learns from positive examples only.

6.3.2 Game evaluation

For the two games studied, Noughts-and-Crosses and Hexapawn, an optimal opponent can always ensure a draw from the initial board state, which leaves no opportunities for the learner to win against an optimal opponent. To ensure possibilities of winning, the game instead starts from a board randomly sampled from the set of one move ahead accessible boards. This set includes all different expected outcomes for the games considered. Then, the actual outcome relies on both the choice of initial board and the sequence of moves performed. We define the Minimax Regret as follows to evaluate the regret of a game given some initial board.

Definition 6.1 (Minimax Regret)**.** *The Minimax Regret of a game is the difference between*

the reward associated with the minimax expected outcome of the initial board and the reward associated with the actual outcome of the game.

The regret represents the loss between the optimal sequence of decisions and an algorithm's sequence of decisions. In the following, we evaluate the learning performance using the Cumulative Minimax Regret:

Definition 6.2 (Cumulative Minimax Regret)**.** *The Cumulative Minimax Regret is the sum of the Minimax Regrets accumulated over a sequence of games.*

The Cumulative Minimax Regret defined in Definition 6.2 does not rely on the choice of initial board and thus provides an absolute measure to evaluate the learning performance of a learning algorithm. It allows to fairly compare several learning systems. Thereafter, we evaluate and compare learning systems using the Cumulative Minimax Regret. The minimax expected outcome of a board can be evaluated from a minimax database computed beforehand. The outcomes won, drawn and lost are respectively associated with the reward values 2, 1 and 0. Therefore, the Minimax Regret ranges between 0 and 2 and the Cumulative Minimax Regret takes positive values.

6.3.3 MIGO algorithm

We present within this subsection details of the algorithm *MIGO*.

MIL *MIGO* is a MIL system [Muggleton *et al.*, 2014; Muggleton *et al.*, 2015]. It takes as input some background knowledge and meta-rules. The background knowledge is a logic program encoding the rules of the game. The meta-rules encode the language bias. Examples are identified from Theorems 6.1 and 6.2 whilst playing. *MIGO* learns first-order logic programs S_W and S_D representing strategies for the task of winning and drawing respectively. *MIGO* is based upon the MIL learning system *Metagol*.

Learning from positive examples Theorems 6.1 and 6.2 assign positive labels to moves for the tasks of winning and drawing. However, these theorems do not allow identifying negative examples for any of these tasks. Therefore, learning is based on positive examples only. *MIGO*, as a MIL system, can make use of language bias to restrict the hypothesis space and generalise from few examples only [Lin *et al.*, 2014; Dai *et al.*, 2017]. However, there is a risk of over-generalisation due to the absence of negative examples.

Dependent Learning *MIGO* learns a global strategy by decomposing the game into a series of inter-related problems. We measure the depth of a board as the number of full moves until the end of the game sequence. The learning starts from lower depth, that is from end-game moves which represent simpler tasks and progressively extends to larger depth, that is to opening moves which are more complex tasks. For successive values of the depth k a series of inter-related definitions are learned for predicates win_$k(A, B)$ and draw_$k(A, B)$. These predicates define maintenance of minimax win and draw in k moves when moving from state A to B. Similarly to Dependent Learning [Lin *et al.*, 2014], the idea is to first learn low-level predicates. The definitions are added into the background knowledge such that they can be used in further definitions. The process continues until the maximum depth is reached.

One-Shot learning *MIGO* is provided with background knowledge representing the rules of the game. To build invented predicates representing new board features and new relational concepts which are not provided in the background knowledge, *MIGO* performs one-shot learning before learning a definition for each depth. *MIGO* learns definitions for each example treated independently with Dependent Learning [Lin *et al.*, 2014]. *MIGO* first sets a limiting complexity. In the following, the complexity is measured as the number of clauses in a logic program. For each single positive example, *MIGO* attempts to construct a program with limited complexity. Whenever a definition is learned, other positive examples which are covered by this definition are removed. When all examples have either been attempted or covered, learned programs are added to the background knowledge as new predicate symbols such that they can be reused when building further definitions. Moreover, the limiting complexity is incremented.

Algorithm 6.1 *MIGO* Algorithm

Input: Positive examples for each *win_k* and *draw_k*, background knowledge B
Output: Strategy S_W for winning and S_D for drawing

```
for k = 1 to Depth do
    one shot learn a rule for each example of win_k/2 and add it to B and to S_W
    Learn win_k/2 and add it to B and to S_W
end for
for k = 1 to Depth do
    one shot learn a rule for each example of draw_k/2 and add it to B and to S_D
    Learn draw_k/2 and add it to B and to S_D
end for
return S_W and S_D
```

Then, *MIGO* considers again examples which still have not been covered by any programs and attempts to learn programs within the augmented complexity bound and while being allowed to reuse previously invented definitions. This process is repeated until all positive examples are covered or until the maximum complexity is reached. Performing one-shot learning before the main learning tasks helps to generate and make available new predicate definitions.

Learning Algorithm The learning algorithm of one iteration of *MIGO* is presented in Algorithm 6.1. Positive examples have been identified following Theorems 6.1 and 6.2. Learning operates in a staged fashion: for increasing values of the depth k, a predicate definition is built while being allowed to reuse previously invented predicates. At each depth, *MIGO* first learns definitions from single examples and with limited complexity which generates reusable invented predicates. Then, *MIGO* learns a definition for the depth considered. In this way, *MIGO* starts by learning simpler rules, and builds more complex rules on top of each other. Each action 'learn' represents a call to the MIL system *Metagol*.

Mixed Learning and Separated Learning Theorem 6.2 assigns positive labels to *draw/2* examples assuming a winning strategy S_W has already been learned. In practice, we distinguish two variants of *MIGO* which deal differently with the availability of a winning strategy S_W:

Separated Learning: *win/2* and *draw/2* are learned in two stages. First, a strategy S_W for *win/2* is learned. A counter measures the number of iterations during which the current

winning strategy S_W has been stable. When this counter exceeds a given threshold value, the learner starts learning a strategy S_D for *draw/2*.

Mixed Learning: *win/2* and *draw/2* are learned simultaneously. Examples of *draw/2* are evaluated with the current strategy S_W for *win/2*. When this latter is updated, examples of *draw/2* are re-tested against the new version of S_W.

Algorithm 6.1 represents the learning process in one iteration. A call to Algorithm 6.1 is made after a game terminates and more examples are identified. However, mixed and S separated learning are learning scheme that span over multiple iterations. Therefore, the difference between mixed and separated learning is not visible in Algorithm 6.1.

6.4 Implementation

6.4.1 Representation

Learned strategies apply to board states. States are atoms of the form $s(B, M)$ and represent the current board B together with the current player M. A board B is encoded as a vector of marks. For instance, for Noughts-and-Crosses, board vectors have size 9, marks are elements from the set $\{O, X, Empty\}$ and the current player is either O or X. For Hexapawn, board vectors have size 9 or 16, marks are elements from the set $\{B, W, Empty\}$ and the current player is either B (black) or W (white).

6.4.2 Primitives and Meta-rules

Learned programs are formed of dyadic predicates, representing actions, and monadic predicates, representing fluents. The language belongs to the language class H_2^2. The background knowledge contains a general move generator *move/2*, which is an action that modifies a state $s(B, M)$ by executing a legal move on board B and updating the current player M. The predicate *move/2* only holds for valid moves: in other words, the learner knows the rules of

the game. The background knowledge also contains two fluents: a won classifier *won/1* and a drawn classifier *drawn/1* which hold when a state is respectively won or drawn. These primitives encode the minimal information a player should know when playing a game. The learner

Name	**Meta-rule**
postcon	$P(a,b) \leftarrow Q(a,b), R(b).$
negation	$P(a,b) \leftarrow Q(a,b), not(R(b,c)).$

Table 6.1: Meta-rules used in *MIGO*: the letters P, Q and R denote existentially quantified second-order variables and the letters a, b, c universally quantified first-order variables.

is provided with the meta-rules *postcon* and *negation* represented in Table 6.1. The *postcon* meta-rule is a standard meta-rule. The meta-rule *negation* is a variant of the *postcon* meta-rule which includes logical negation for primitive predicates. It is implemented as negation as failure. This meta-rule allows for the negation of primitive predicates without having to include each negated primitives as a new primitive. However, this form of negation does not allow for the introduction of negated invented predicates and only can negate existing primitive predicates. We overcome this limitation with the use of Dependent Learning and one-shot learning. Dependent Learning and one-shot learning allow the construction of invented predicates which are successively added as new primitive predicates. Thus, the learner can negate predicates invented in previous depths.

6.4.3 Execution of the strategy

The learner plays following its learned strategy. For increasing values of k, a clause of the form '`win(A,B) :- win_k(A,B).`' is added to the background knowledge. Then, clauses of the form '`draw(A,B) :- draw_k(A,B).`' similarly are added to the background knowledge for increasing values of k. When executing a strategy described with an hypothesis H, the move performed is the first one consistent with H. Practically, *MIGO* first attempts to execute *win/2* and attempts to prove *win_k/2* for increasing values of k. Failing that, it attempts to execute *draw/2* and attempts to prove *draw_k/2* for increasing values of k. If all these proofs fail, a move is selected at random among legal moves. Applying winning rules before drawing rules, and each in increasing depth order, can be justified as the choice of the action

with maximum expected reward, such as choosing the action with the maximum Q-value in reinforcement learning.

The opponent chosen is an optimal player following the minimax algorithm to comply with Assumption 6.1. The opponent plays a deterministic minimax strategy that yields the best outcome in the minimum number of moves. Optimal moves are identified from a database computed beforehand.

6.4.4 Learning a strategy

At the end of a game, the outcome is observed and the sequence of visited states is divided into moves. The depth of each move is measured as the number of full moves until the end of the game in the observed sequence. Moves are added to the training set for *win_k/2* or *draw_k/2* if they satisfy Theorems 6.1 or 6.2 respectively. Strategies are relearned from scratch after each game using Algorithm 6.1. One additional constraint is added such that *draw/2* cannot be learned before a strategy for *win/2* exists since this would cause the learner to always draw and never win.

6.5 Experiments

6.5.1 Experimental Hypotheses

This Section describes experiments which evaluate the performance of *MIGO* for the task of learning optimal two-player game strategies[2]. To support our claim that *MIGO* achieves improved sample efficiency compared to reinforcement learning systems, we first investigate the following Experimental Hypothesis:

Experimental Hypothesis 6.1. *MIGO can converge faster toward optimal two-player game strategies than state-of-the-art reinforcement learning systems.*

[2]The code for reproducing the experiments is available at `https://github.com/migo19/migo.git`

To test this Experimental Hypothesis, we measure the speed of convergence with the Cumulative Minimax Regret defined in Definition 6.2. The Cumulative Minimax Regret is an absolute measure which allows to evaluate and compare the performance of learning systems. Therefore, our experiments consider evaluable games for which we can evaluate the Cumulative Minimax Regret. Specifically, our experiments consider the task of learning strategies for the games of Noughts-and-Crosses and a variant of the game of Hexapawn [Gardner, 1962]. We compare *MIGO* against several representative reinforcement learning systems: MENACE / HER, Q-learning and Deep Q-learning. While MENACE [Michie, 1963] and HER [Gardner, 1962] are specifically designed to learn optimal strategies for Noughts-and-Crosses and Hexapawn respectively, Q-learning and Deep Q-learning are general reinforcement learning algorithms, which in particular are applicable to both games studied. Moreover, we assume an optimal opponent. This particular setting falls within the limits of the assumptions of all systems evaluated. All systems are evaluated under the same experimental conditions. Therefore, the only variable is the learning system. At the end, we investigate the following Null Hypothesis associated with the previous Experimental Hypothesis 6.1:

Null Hypothesis 6.1. *MIGO cannot converge faster in terms of Cumulative Minimax Regret than MENACE / HER, Q-learning and Deep Q-learning for learning optimal two-player game strategies.*

We additionally test the ability of *MIGO* to transfer learned strategies to more complex games. Our second experiment evaluates whether knowledge learned in a source task can be leveraged to speed up learning in a related subsequent target task. Thus, we test the following Experimental Hypothesis:

Experimental Hypothesis 6.2. *MIGO supports transfer learning of learned strategy.*

Similarly, we will evaluate the performance using the Cumulative Minimax Regret. Our experiment will evaluate the capacity to transfer knowledge in both directions between Hexapawn and Noughts-and-Crosses. We associate to this Experimental Hypothesis the following Null Hypothesis:

Null Hypothesis 6.2. *MIGO cannot converge faster in terms of Cumulative Minimax Regret when transferring the strategy learned for a game to a different game.*

6.5.2 Convergence

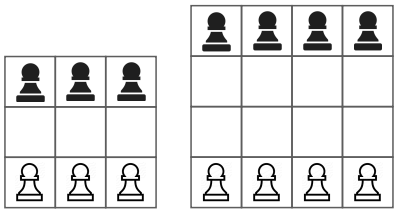


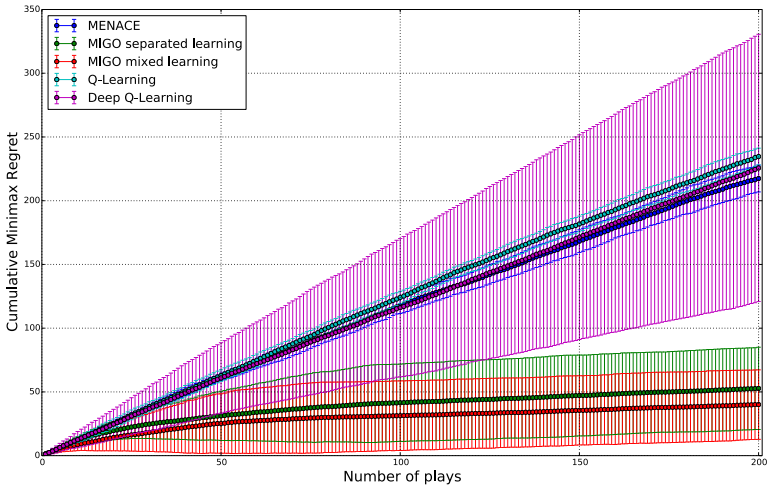
Figure 6.2: Initial boards for Hexapawn_3 and Hexapawn_4

Materials The first game we consider is Noughts-and-Crosses. A board of Noughts-and-Crosses is represented in Figure 6.1a. We recall games start from an initial board sampled from the set of one-move ahead boards. In Noughts-and-Crosses, the set of one-move ahead boards comprises 12 boards after taking into account rotations and symmetries. Among them 7 are expected win, and 5 are expected draw. Therefore the expected worst case regret is 1.58 for Noughts-and-Crosses.

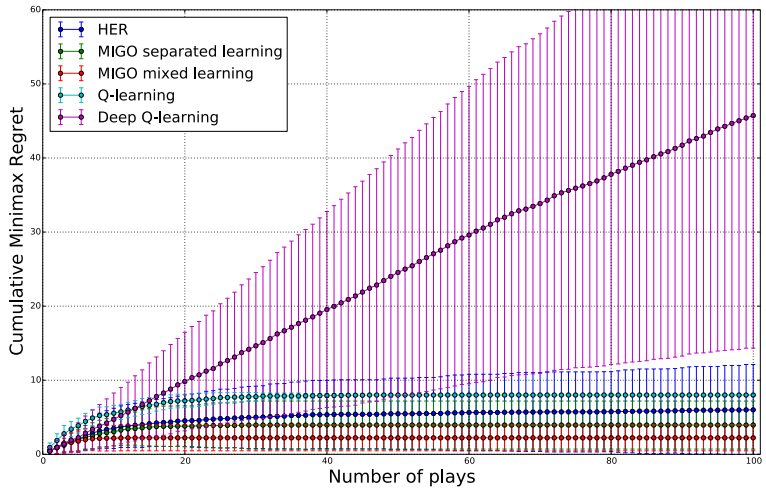
The second game we consider is Hexapawn. Hexapawn's initial board is represented in Figure 6.2. In this game, the goal of each player is to advance one of their pawns to the opposite end. Pawns can move one square forward if the next square is empty or capture another pawn one square diagonally ahead of it. We have modified the rules to include the possibility of draw: the game is said to be drawn when the current player has no legal move. Thereafter, we refer to Hexapawn_3 and Hexapawn_4 for the game of Hexapawn in dimensions 3 by 3 and 4 by 4 respectively. For Hexapawn_3, the set of one-move ahead positions comprises 5 boards after taking into account the vertical symmetry. Among them, 3 are expected draw and 2 are expected win. Therefore the expected worst case regret is 1.4. For Hexapawn_4, the set of one-move ahead positions comprises 8 unique boards, 5 of which are expected win, 2 expected draw and 1 expected loss. Therefore the expected worst case regret is 1.5.

Methods We evaluate all learning systems within the same experimental conditions. All systems play games starting from the same set of initial boards randomly sampled from the set of one-full-move-ahead positions. All systems evaluated face the same deterministic minimax player. The learner always starts the game. We follow an implementation of Tabular Q-learning available from [fheisler, 2015]. This implementation of Tabular Q-learning has specifically been designed and tuned for the game of Noughts-and-Crosses. We used the parameter values provided: the exploration rate is set to 0; the initial q-values are 1; the discount factor is $\gamma = 0.9$ and the learning rate $\alpha = 0.3$. Similarly, we follow an implementation of Deep Q-learning available from [yanji84, 2016]. This implementation has specifically been designed and tuned for the game of Noughts-and-Crosses. We use the parameter values provided: the discount factor is set to 0.8; the regularisation strength to 0.01; the target network update rate to 0.01; the initial and final exploration rate are 0.6 and 0.1 respectively; the batch size is 32. We implemented MENACE and HER following the description and parameters from [Michie, 1963] and [Gardner, 1962]: one-move-ahead states initially contain 3 beads of each colour, two-move-ahead states 2 beads and three-move-ahead states 1 bead. A win is rewarded with three additional beads, a draw with one additional bead and a loss is punished with one bead forfeit. Regarding *MIGO*, in separated learning, the threshold value over the counter for starting learning *draw/2* is set to 10 for Noughts-and-Crosses and to 5 for Hexapawn$_3$ as the dimensions are smaller.

Results Results have been averaged over 20 runs for Noughts-and-Crosses and 40 runs for Hexapawn$_3$. Results are presented in Figure 6.3 and show that both variants of *MIGO* converge significantly faster than MENACE / HER, Q-learning and Deep Q-learning for both games. Both variants of *MIGO* require around 60 games to converge for Noughts-and-Crosses while MENACE, Q-learning and Deep Q-learning need more than 200. *MIGO* requires around 12 games to converge for Hexapawn$_3$ while HER needs around 38, Q-learning around 31 and Deep Q-learning more than 100. These results evidence two cases for which *MIGO* outperforms the reinforcement learning systems tested. Therefore, we can refute Null Hypothesis 6.1. MENACE, HER and Q-learning encode the knowledge into parameters (number of beads or Q-values)



(a) Noughts-and-Crosses



(b) Hexapawn$_3$

Figure 6.3: Cumulative Minimax Regret versus the number of iterations for Noughts-and-Crosses and Hexapawn$_3$

which are unique for each action from each state. Knowledge cannot be transferred from one state to another which results in a weaker generalisation ability. Deep Q-learning can provide some generalisation ability; however, it is only visible after a large number of iterations. Conversely, *MIGO* generalises the board characteristics and each rule learned describes a set of states, which considerably reduces the number of parameters to learn and therefore reduces the number of examples required for convergence. The maximum depth is larger for Noughts-and-Crosses than for Hexapawn$_3$. Therefore, all systems require more iterations to converge for Noughts-and-Crosses compared to Hexapawn$_3$.

The systems presented have different ranges of applications. MENACE and HER are specifi-

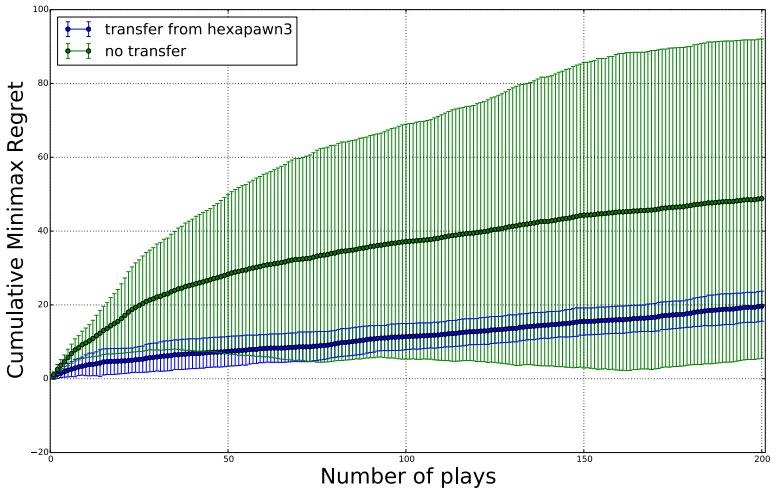
cally tailored for Noughts-and-Crosses and Hexapawn_3 respectively. Conversely, Q-learning and Deep Q-learning are a general approaches that can tackle a wide range of tasks providing that parameters are tuned. For instance, deep Q-learning performs worst on Hexapawn_3 than other systems. This can be explained by the fact that the parameters selected are the ones tuned for Noughts-and-Crosses and might not be adapted to a different game. Finally, *MIGO* assumes an optimal opponent which reduce its range of applications to the set of evaluable games.

The systems evaluated have different learning strategy. MENACE and HER have a non-deterministic strategy, these systems assign a probability for each possible move which is revised during learning. Q-learning and deep Q-learning explore with some small probability in which case an action is chosen at random, and they choose the best option as the one with the highest Q-value otherwise. Conversely, *MIGO* plays the first consistent move while executing the strategy. If no moves are consistent with the current strategy, a move is selected at random, which corresponds to an exploration strategy. While *MIGO* uses a formalised logical bias in the form of meta-rules and encoded in the search procedure, MENACE, HER, Q-learning and deep Q-learning encode bias into hyper-parameters values and into their model structure.

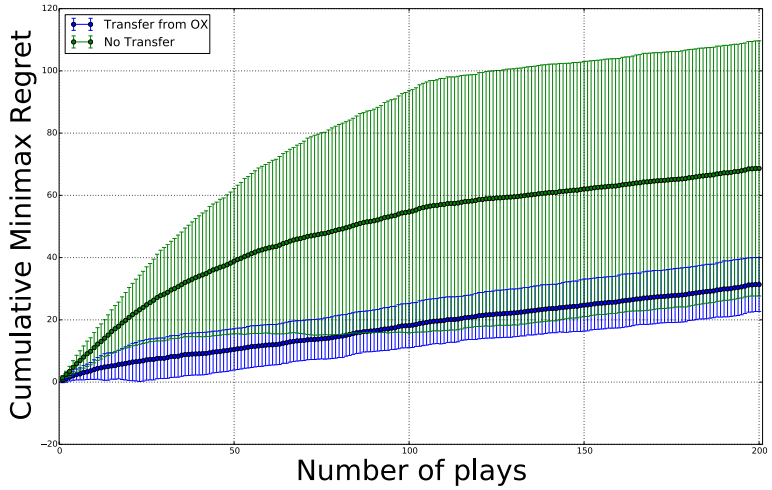
For both games, the version of *MIGO* with mixed learning has lower Cumulative Minimax Regret than separated learning. This can be explained by the fact that mixed learning does not waste any examples of *draw/2* from the initial period in which *win/2* is being learned and it does not stop learning *win/2* after the initial period.

6.5.3 Transferability

Materials and Methods Strategies are first learned for the source tasks Hexapawn_3 and Noughts-and-Crosses following the same method as in the previous experiment. Strategies are learned with mixed learning for 100 iterations for Hexapawn_3 and 200 iterations for Noughts-and-Crosses. The resulting learned program is transferred to the target learning task Noughts-and-Crosses and Hexapawn_4 respectively. Only the background knowledge changes from one task to another, which is the definitions of *move/2*, *won/1* and *drawn/1*. Results have been averaged over 20 runs.



(a) Hexapawn$_3$ to Noughts-and-Crosses



(b) Noughts-and-Crosses to Hexapawn$_4$

Figure 6.4: Transfer Learning Experiment: Results. Similar results are obtained from Hexapawn$_3$ to Hexapawn$_4$.

Results Results are presented in Figure 6.4. Results show that transferring the strategy learned for a previous game to a different game helps to converge faster. The number of games until convergence when learning Noughts-and-Crosses drops from 60 to 15 if being trained first on Hexapawn$_3$. The number of games until convergence when learning Hexapawn$_4$ drops from 110 to 25 if being trained first on Noughts-and-Crosses. The learner benefits from the knowledge transferred and its performance accordingly are substantially improved compared to an initial random player. Therefore, we refute Null Hypothesis 6.2. This experiment demonstrates learned strategies are general enough to be applicable in multiple contexts in a straightforward way. *MIGO* learns first-order rules which are very general and that can be applied to a multitude of different games, which is not true of reinforcement learning systems. Although this

experiment considers games which are relatively similar, these results exhibit *MIGO*'s inherent capacity for transfer learning.

6.5.4 Comprehensibility

Depth	Rule
1	`win_1(A,B):-win_1_1_1(A,B),won(B).` `win_1_1_1(A,B):-move(A,B),won(B).`
	`draw_1(A,B):-draw_1_1_3(A,B),not(win_1(B,C)).` `draw_1_1_3(A,B):-move(A,B),not(win_1(B,C)).`
2	`win_2(A,B):-win_2_1_1(A,B),not(win_2_1_1(B,C)).` `win_2_1_1(A,B):-move(A,B),not(win_1(B,C)).`
	`draw_2(A,B):-draw_2_1_1(A,B),not(win_1(B,C)).` `draw_2_1_1(A,B):-draw_1(A,B),not(win_1(B,C)).`
3	`win_3(A,B):-win_3_1_1(A,B),not(win_3_1_1(B,C)).` `win_3_1_1(A,B):-win_2_1_1(A,B),not(win_2(B,C)).`
	`draw_3(A,B):-draw_3_1_10(A,B),not(draw_1_1_3(B,C)).` `draw_3_1_10(A,B):-draw_2(A,B),not(draw_1_1_3(B,C)).`
4	`draw_4(A,B):-draw_4_1_2(A,B),not(draw_1_1_3(B,C)).` `draw_4_1_2(A,B):-draw_3(A,B),not(draw_1_1_3(B,C)).`

Table 6.2: Example of rules learned for Noughts-and-Crosses (all) and Hexapawn$_3$ (above the double line)

Rules learned by *MIGO* are presented in Table 6.2. *MIGO* converges toward this full set of rules when playing Noughts-and-Crosses. Because the maximum depth of Hexapawn$_3$ is smaller, *MIGO* learns rules up to the double line when playing Hexapawn$_3$. After unfolding, the first rule can be translated into English as: *State A is won at depth* 1 *if there exists a move from A to B such that B is won.* Similarly, winning at depth 2 can be described with the following statement: *State A is won at depth* 2 *if there exists a move of the current player from A to B such that B is not immediately won for the opponent and such that the opponent cannot make a move from B to C to prevent the current player from immediately winning.* This statement is similar to the one presented in Section 6.1. Finally, winning at depth 3 can be explained as: *State A is won at depth* 3 *for the current player if there exists a move from A to B such that B is not won for the opponent in* 1 *nor* 2 *moves and such that the opponent cannot make a move from B to C to prevent the current player from winning in* 1 *or* 2 *moves.* The use of a literal followed by its immediate negation means that the current player has some advantage

that the opponent has not. Since the current player changes with the board state, the first literal applies to a state for which it is one player to move, and the second applies to a state for which it is its opponent's turn. The same last literal is repeated in depth 1 rules because of the language bias.

Rules are hierarchically built on top on each other. The calling diagram in Figure 6.5 represents the dependencies between each learned predicates. Lower depths are at the bottom of the calling diagram while higher depths are at the top. The strategy learned can be interpreted as an iterative deepening search: it is a form of minimax search which looks ahead until the completion of the game. This interpretation of the learned strategy supports results from Subsection 6.5.3: a minimax strategy is a general strategy which can be applied to a wide range of two-player games. What is interesting is that a same system can rediscover a usual and effective strategy that can be applied to different games, which is not true of reinforcement learning systems learned models.

Even though this learned strategy is complex to understand due to the presence of negations and dependencies, especially for larger depths, it offers insights which allow to understand its execution. By opposition, none of the reinforcement learning systems studied can provide similar explanation of the moves chosen. The reinforcement learning systems tested have an implicit representation of the problem and their learned strategy is encoded into parameter values. In this sense, their meaning is more obscure and difficult to decrypt. Moreover, they do not rely on background knowledge. For instance, no geometrical concepts have been encoded. Conversely, *MIGO* benefits from background knowledge which describes the notion of winning, and from which it extracts a concept of alignment. The fact that the strategy is based on background knowledge allows some degree of explanability. Still, the background knowledge provided is abstract enough which allows playing a wide range of games and supports transfer learning.

We refer to Michie's criteria described in Section 2.3.9 for evaluating the performance of Machine Learning systems. *MIGO*'s predictive performance augments with increasing amount of data and learned hypotheses are provided in symbolic form. Therefore, *MIGO* verifies the *weak* and

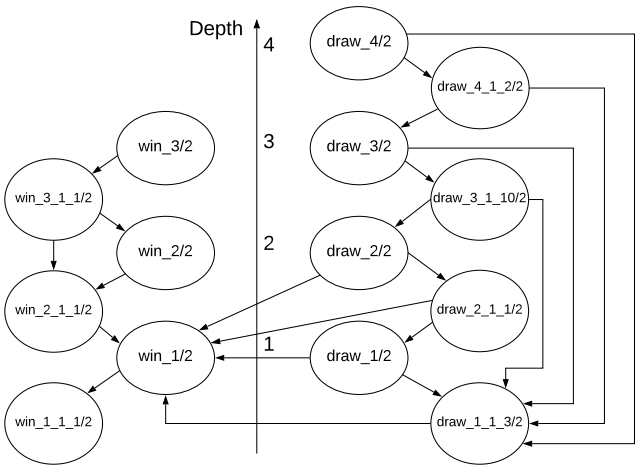


Figure 6.5: Calling diagram of Learned Strategies

strong criteria. It remains to evaluate whether *MIGO* can fulfil the *ultra-strong* criterion. It is first notable that the rules shown in Table 6.2 provide a certain form of explanation. Although larger depths are more complex to understand, all rules can be translated into English as a chain of relations and features explaining the decisions. These insights are a significant advantage compared to reinforcement learning approaches which can not provide similar understandable explanations. To test the *ultra-strong* criterion, an operational definition of comprehensibility of hypotheses has been proposed [Muggleton *et al.*, 2018b]. This operational measure is based on empirical tests involving human participants. Experimental participants are presented with explanation based on a logic program hypothesis and are given time to study it. Their degree of comprehension of this hypothesis is assessed as the mean accuracy with which a participant can classify new material sampled randomly from the hypothesis's domain. The explanatory effect is evaluated as the impact of explanations on human comprehensibility: it is the difference in comprehensibility between a human population aided by machine learned explanation and a human population studying the data alone.

A variant of *MIGO* has been tested for the *ultra-strong* criterion [Ai *et al.*, 2021]. Results indicate that machine learned explanations can facilitate human understanding and learning. In some circumstances, human learning aided by a symbolic machine learned theory achieves significantly higher performance than the self-learning human population. For instance, being provided with machine learned explanations improves textual answer quality for depth 2 of human participants which indicates that explanations have a beneficial explanatory effect in

	Hexapawn_3	OX	Hexapawn_4
MIGO mixed learning	$3.0.10^{-3}$	$1.5.10^{-1}$	3.9
MIGO separated learning	$2.8.10^{-3}$	$8.9.10^{-2}$	3.8
MENACE / HER	$2.7.10^{-4}$	$1.5.10^{-3}$	/
Q-Learning	$1.9\ .10^{-3}$	$2.3.10^{-1}$	$2.7\ .10^{-1}$
Deep Q-Learning	$1.7.10^{-2}$	$2.4.10^{-1}$	$2.1\ .10^{-1}$

Table 6.3: Average CPU time (seconds) of one iteration: it includes learning a strategy and executing a move from this strategy.

this case. However, machine learned explanations lead to degradation of human performance when explanations are not of appropriate complexity. For instance, participants in the machine-aided group have reduced performance for depth 3 questions after receiving explanations based on the learned *win_3/2* predicate compared to the self-learning group. Finally, providing explanations of a theory which does not reduce the executional cognitive cost neither does lead to a beneficial explanatory effect. For instance, no significant beneficial explanatory effects has been observed for depth 1 since the learned strategy does not provide any helpful executional short-cut.

6.5.5 Running Times

Average running times of one iteration are presented in Table 6.3 for the experiment presented in Subsection 6.5.2. Games are ordered by increasing complexity measured as the maximum depth. These results show running time increases rapidly with the game complexity for *MIGO*. This reflects the increasing execution time of the learned strategy which is not efficient. *MIGO*'s learned strategy conducts a deep evaluation which requires a complete search through the corresponding game sub-tree to decide whether a move leads to a win. This strategy is equivalent to a minimax search. Therefore, the time complexity is a function of the number of possible moves and the depth of the game tree, which limits scalability. In particular, it prevents transferring this strategy to non-evaluable games in which computing minimax evaluation in intractable. This learned strategy is applicable to evaluable games, which is the domain *MIGO* was trained on. Future work is needed for *MIGO* to learn strategies for non-evaluable games. Although running time of one iteration is longer for *MIGO*, cumulative running time to reach convergence

is smaller for *MIGO* compared to Q-learning and Deep Q-learning, for Noughts-and-Crosses and Hexapawn$_3$ and of the same order than HER for Hexapawn$_3$ since *MIGO* requires fewer examples to converge. Follow-up work has evaluated a variant of *MIGO* which optimises the execution time for hypothesised programs [Ai *et al.*, 2021]. Selection of hypotheses is based on the efficiency of hypothesised programs and is performed using Metaopt [Cropper and Muggleton, 2018]. The efficiency of hypotheses is evaluated as the tree cost which measures the size of the SLD-tree searched when a program is given a goal. In addition, the strategy learned by this variant of *MIGO* uses heuristic short-cuts which helps efficiently pruning branches in the game tree. An example of a heuristic is the additional primitive *number_of_pairs/3*. This primitive depicts the number of pairs of a player on a given state of Noughts-and-Crosses.

6.6 Future Work

We identify the following limitations of these contributions which could be addressed as future work.

More complex games We have restricted the scope of this work to evaluable games. Future work needs to be conducted to scale up *MIGO* to learn strategies for non-evaluable games. A first limitation to scalability is the restriction imposed by the initial Assumption 6.1: the current version of *MIGO* requires an optimal opponent, which is intractable in large dimensions. Addressing this issue will require to relax Theorems 6.1 and 6.2. A solution could be to learn from repeated self-play, in which the algorithm learns by playing against successive versions of itself and thus does not require an expert player as opponent.

Another limitation to scalability is the complexity dependence on the depth of the game. We have observed in our experiments that the learning time increases with the depth of the game due to increasing execution time of the learned strategies. The learned strategy conducts a form of minimax search whereas checking all possible paths in a game tree is inefficient for complex problems. Alternatively, we suggest complex games can be broken up into different

sub-games corresponding to the achievement of sub-goals identified with intermediate rewards. The learning will include ways of combining sub-games strategies into a top-level strategy. Moreover, hypotheses can be preferred depending on their efficiency [Cropper and Muggleton, 2018] and the search can be improved incorporating heuristic tests [Ai *et al.*, 2021].

Our experimental results have evaluated the performance of *MIGO* over two evaluable games and have demonstrated its strengths. Future work is needed to test our approach on a wider range of games. For instance, the General Game Playing framework [Genesereth *et al.*, 2005; Genesereth and Thielscher, 2014] contains hundreds of games with great variety. This dataset includes different types of games with varying number of players, complexity, games with complete or partial information, simultaneous or alternating play and thus this dataset provides a relevant and challenging and relevant test-bed.

Over-generalisation One limitation of the learning system presented is the risk of over-generalisation due to the lack of negative examples. The learned strategy presented is not exactly optimal and the Cumulative Minimax Regret derivative does not exactly converge to the optimal value 0. As future work, the implementation could be extended to include a more thorough context for learning from positive examples only [Muggleton, 1996]. An alternative is to extend our credit assignment protocol portrayed by Theorem 6.1 and 6.2 to also identify negative examples for the tasks of winning and drawing.

Transfer Learning We have experimentally demonstrated in Chapter 6 that *MIGO* can transfer the strategy learned in a simple domain to a more complex one. Transferring the strategy from a different domain was beneficial in this scenario and led to faster convergence in the target domain. However, the games we have considered share a similar winning strategy. Future work will need to be conducted to evaluate whether transfer learning is possible between more distant domains for which only part of the transferred strategy is useful in the target domain. The objective is to test for possible negative transfer effects resulting from the transfer of imperfect, incomplete or irrelevant concepts. We believe *MIGO* can leverage the hierarchical decomposition of learned strategies to retain and reuse only useful invented predicates and

forget irrelevant ones. In this case, the knowledge transfer would only change the bias. New tasks would be solved using concepts from several previously encountered tasks combined in a novel way. A related direction for future work is the revision of the strategy learned in a first domain to fit the target domain. Such revision methods could be based on the ABC Repair system [Li *et al.*, 2018].

Despite these limitations, we believe our contributions introduced in this Chapter open encouraging opportunities for machine learning game strategies.

6.7 Summary

This Chapter has introduced a novel logical system named *MIGO* for learning optimal strategies for two-player evaluable games. *MIGO* is a MIL learner. *MIGO* learns from playing, its actions affect the subsequent data it receives and thus its current strategy shapes the instance space. We have claimed and demonstrated that *MIGO* achieves lower sample complexity and improved learning performance compared to reinforcement learning systems and that *MIGO* addresses a number of drawbacks inherent in current reinforcement learning systems.

First, our experiments have demonstrated that *MIGO* achieves significantly lower Cumulative Minimax Regret compared to Deep and classical Q-Learning and that *MIGO* outperforms these systems in terms of sample efficiency. *MIGO* can generalise over board states and each rule learned is applicable to a range of board states. Conversely, Q-learning identifies parameters which are unique for each action from each state and accordingly require very large training sets to converge. Deep Learning can generalise over states but inherently requires large training sets. Moreover, *MIGO* is based on Dependent Learning and can reuse rules acquired for smaller depths to build more complex rules. *MIGO* can make use of similarities between situations and combine previously learned knowledge accordingly. In this sense, Dependent Learning helps to shape the hypothesis space to guide learning. It introduces a bias toward previously used predicates which in turn improves the learning efficiency.

Moreover, we have demonstrated that strategies learned with *MIGO* are transferable to more

complex games with significantly less retraining. *MIGO* learns rules which are general enough to be applicable to a variety of games. Conversely, the reinforcement learning systems tested are brittle. They lack the ability to reason on an abstract level which prevent them to perform transfer learning, even between very similar tasks.

Finally, strategies learned with *MIGO* have been shown to provide some form of explanability. Moves chosen can be associated with a human comprehensible description of relations and features tracing the reasons that led to this decision. Conversely, the reinforcement learning systems evaluated operate in a largely opaque way. We believe having access to some form of explanation is a step toward greater transparency, essential in domains in which trustworthiness and verifiability are necessary, and therefore constitutes a considerable advantage compared to reinforcement learning systems.

This Chapter has demonstrated a method to revise both the instance space and the hypothesis space to improve the sample complexity and learning performance in MIL, thus supporting Subthesis S.3. The next Chapter 7 summarises the contributions presented in this thesis, concludes this thesis and discusses directions for future work.

Chapter 7

Conclusions and Future Work

In this Chapter, we conclude this thesis, review our contributions introduced in the previous chapters and discuss directions for future work.

7.1 Conclusions

7.1.1 Motivation and Claims

This thesis is concerned with the induction of hypotheses, as logic programs, from examples. Induction is a prime ability of human intelligence and is the ambition of Machine Learning. However, induction is inherently complex due to large search spaces. ILP is a form of Machine Learning for inducing logic program hypotheses generalising specific observations. ILP benefits from the high expressivity of logical representation languages. Moreover, ILP systems can make use of inductive bias and background knowledge which provides high data-efficiency. Finally, learned models are expressed in logical form which facilitates comprehensibility. We have focused more specifically in this thesis on MIL, which is a state-of-the-art form of ILP. MIL supports predicate invention, learning of complex theories involving recursions, first-order and higher-order definitions. However, current MIL approaches have limited efficiency: MIL has a sample complexity polynomial in the size of learned programs and a learning complexity

exponential in the size of learned programs. We have investigated in this thesis revision methods to achieve more efficient MIL. We have claimed that improvements over the sample and learning complexity in MIL can be achieved through:

(Subthesis S.1) methods that revise the instance space,

(Subthesis S.2) methods that revise the hypothesis space and

(Subthesis S.3) methods that revise both the instance space and the hypothesis space.

The instance space is the set of items over which hypotheses are defined. It is the set of possible examples. Subthesis S.1 claims that there exist methods revising the instance space in MIL which can guide the selection toward informative examples. The hypothesis space is the set of hypotheses that can be formulated and may be output by the learner. Subthesis S.2 claims that there exist methods revising the hypothesis space in MIL which can guide the search for consistent hypotheses. Finally, Subthesis S.3 states that there exist methods revising both the instance and hypothesis space in MIL which can improve the sample and learning efficiency in MIL.

7.1.2 Contributions

To support our claims, we have introduced one method for each of the categories above. We have theoretically and experimentally evaluated their sample and learning efficiency. Specifically, we have made the following contributions in theory, methods, implementation and applications of MIL.

We have presented in Chapter 4 a method which revises the instance space with active learning in MIL and aimed at learning agent strategies. The learner chooses experiments and queries their label to an oracle. We have demonstrated this method can require significantly less labelled data to converge toward agent strategies. This method extends Bayesian MIL into Active Bayesian MIL. An Active Bayesian MIL learner allocates a posterior distribution over the hypothesis space. It samples a set of consistent hypotheses from this posterior distribution

using meta-interpretation. It computes the entropy of candidate instances from this set of sampled hypotheses. Then, it selects an instance with maximum entropy among the set of candidate instances. This instance is maximally informative and discriminative with respect to the remaining competing hypotheses. Therefore, the active learning strategy which consists of the selection of high entropy instances can reduce the sample complexity. We have theoretically demonstrated that the probability of selecting an instance with globally maximal entropy over the instance space for an active learner is N times the one of a passive learner, where N is the number of unlabelled instances available. We have provided and described an implementation of Active Bayesian MIL. We have experimentally demonstrated the number of experiments to perform to reach an arbitrary accuracy level can at least be halved with Active Bayesian MIL compared to Passive Bayesian MIL when learning agent strategies. These contributions support Subthesis S.1.

We have introduced in Chapter 5 a method to revise the hypothesis space in MIL based on predicate invention. The learner generates in a pre-processing step a set of invented predicates. These predicates can be reused during the construction of a consistent hypothesis. Owing to the reuse of these invented predicates, the target hypothesis has fewer clauses and therefore is easier to learn. These invented predicates are generated bottom-up from the background knowledge. In MIL, the background knowledge contains meta-rules which are second-order clauses. This second-order program background knowledge is generalised with an extension of the immediate consequence operator for second-order logic programs. The learner attempts to resolve body literals in meta-rules using the current background knowledge. For each successful resolution, a Skolem constant is generated and is bound to a potential second-order variable in the meta-rule head. This Skolem constant represents a new predicate symbol which definition is given by the associated meta-substitution. This definition is saved in the background knowledge and this process is iterated. The learner thus produces a set of invented predicates which can be reused when constructing a consistent hypothesis. We have defined an equivalence relation between predicates based on equivalence of success sets and we have used it for eliminating redundant predicates. We have theoretically derived an upper bound over the number of new predicates symbols introduced. We have theoretically demonstrated our bottom-up method

is complete with respect to a fragment of dyadic Datalog. We have provided and described an implementation of our predicate invention method. We have theoretically and experimentally demonstrated this method can improve the learning and sample efficiency in MIL. These contributions support Subthesis S.2.

We have introduced in Chapter 6 a MIL learner called *MIGO* aimed at learning optimal game strategies for two-player games. *MIGO* learns from playing and obtains examples by executing its current strategy. Thus, instance selection is directed toward regions identified with its current strategy. Moreover, *MIGO* uses Dependent Learning to revise its hypothesis space. *MIGO* first learns strategies for smaller depths, and progressively learns winning strategies for larger depths. It thus constructs a hierarchical series of predicates with increasing level of abstraction. Lower-level predicates can be reused when building increasingly complex predicates which promotes learning and sample efficiency. Given sufficient play and when playing against an optimal opponent, *MIGO* learns optimal strategies for evaluable games. One advantage of considering evaluable games is that there is a tractable approach to calculating Minimax Regret, which provides an absolute measure for evaluating and comparing the performance of learning algorithms. We have provided and described an implementation of *MIGO*. We have used evaluable games to compare Cumulative Minimax Regret for variants of reinforcement learning against our system *MIGO*. We have experimentally demonstrated that *MIGO* achieves significantly lower Cumulative Minimax Regret compared to Deep and classical Q-Learning. In addition, we have experimentally demonstrated that strategies learned with *MIGO* are transferable in between different games. Finally, the strategies learned by *MIGO* are comprehensible to humans. These contributions support Subthesis S.3.

7.2 Future Work

We identify the following research directions which could be investigated as future work to address limitations of this thesis.

7.2.1 Revising the instance space

Active Learning Strategy In Chapter 6, we have demonstrated *MIGO* achieves significantly lower sample complexity compared to the reinforcement learning systems tested. *MIGO* chooses its moves by executing its current learned strategy which directs instance selection. Future work is needed to evaluate whether active learning could further help to reduce the sample complexity. An active learner could choose an initial board to start the game from. This choice could be based on an expected information gain criterion.

In Chapter 4, we have presented an active learning query strategy based on an expected information gain criterion evaluated from the entropy. We have demonstrated this query strategy can halve experimental costs. Future work could investigate and compare different active learning query strategies. However, the performance of query strategies is likely to rely on the characteristics of the problems and datasets on hand and may even vary over time as more instances are being labelled. Therefore, an interesting perspective for future work is to learn adequate query strategies as a form of meta-learning instead of relying on pre-set choices. For instance, a master strategy can be identified online as a combination of an ensemble of active learning heuristic strategies with the help of multi-armed bandit algorithms [Baram *et al.*, 2004; Hsu and Lin, 2015]. In [Ebert *et al.*, 2012], the active learning strategy is learnt as a feedback-driven Markov decision process combining exploitation and exploration criteria with adaptive and time-varying trade-off. Alternatively, the active learning strategy can be treated as a regression model for which parameters are learned and which predicts the reduction in generalisation error that can be expected by querying the label of a data point [Konyushkova *et al.*, 2017].

Sampling process In active learning, it is usually assumed that the learner has access to a pool of instances sampled beforehand according to a predefined distribution, as it was the case in Chapter 4. Sampling is essential to overcome potentially infinite instance spaces and for efficiency considerations. For instance, when learning game strategies as in Chapter 6, an active learner could choose a board to start the game from. Since the size of the instance space

increases with the complexity of the game, the choice of initial boards to start the game from may become intractable without sampling. However, as Lemma 4.1 showed, the existence of informative instances among a sampled set of instances relies on the size of the pool. Therefore, there is a trade-off between the existence of highly informative instances in the pool and the efficiency of the search for informative instances within the pool. We suggest future work could be conducted to investigate improvements over the sampling process of instances to balance this trade-off. For instance, future work could study whether adaptive sampling strategies can improve the informativeness of sampled instances and in turn the sample complexity. In [Dasgupta and Hsu, 2008], the system exploits cluster structure in the data to guide sampling toward informative clusters. More generally, adaptive sampling processes could allow navigating the instance space efficiently and informatively. The sampling distribution will be updated at each iteration to incorporate newly acquired information and be representative of the current version space. The sampling distribution will be directed toward regions of interest in the experimental design space. These regions of interest can, for instance, be identified as high surface variance areas in the predictions.

A sampling process suggests the possibility to synthesise instances *de novo* [Angluin, 1988]. A challenge is that experiments generated and invented by the learner need to be practically realisable and their outcome interpretable. However, it has been shown that artificially generated queries are not always interpretable and can not always be labelled reliably by humans oracles [Baum and Lang, 1992].

7.2.2 Revising the hypothesis space

Repairing hypotheses In both Chapters 4 and 6, after a new training instance is added to the training set, new consistent hypotheses are generated from scratch. Alternatively, consistent hypotheses could be generated by repairing hypotheses consistent with the previous training set. Future work needs to be conducted to investigate whether hypothesis repair can provide a more efficient hypothesis generation process. Repairing aims at automatically detecting faults, such as inconsistency, incompatibility or insufficiency, and possible fix to alleviate conflicts with

observations. For instance, belief revision [Gärdenfors and Rott, 1995] includes mechanisms for re-establishing consistency after receiving new information, these mechanisms involve the deletion or modification of sentences. Reformation [Bundy and Mitrovic, 2016] additionally can suggest changes to the language. The ABC repair system [Li *et al.*, 2018] can repair Datalog theories by combining abduction, belief revision and reformation. In general, multiple repairs may be possible and a challenge is to identify optimal repairs [Urbonas *et al.*, 2020]. The identification of repairs of these forms could be investigated as future work to efficiently form new consistent hypotheses.

Selection of relevant predicates We have demonstrated in Chapter 5 that performing bottom-up iterations can result in a sample complexity gain when the condition expressed in Proposition 5.1 is fulfilled. However, this condition depends on the number of predicates introduced. We have proposed a criterion for eliminating redundant predicates but we have not provided a guarantee that it results in the fulfilment of the condition expressed in Proposition 5.1. Similarly, in Chapter 6, the learning system *MIGO* performs Dependent Learning and systematically saves all learned predicates in the background knowledge. Both these situations can lead to a catastrophic remembering problem: the background knowledge monotonically grows due to the inability of the learner to forget knowledge and this can overwhelm the search. Future work is needed to devise a more discriminative selection of relevant predicates. In Chapter 5, we have proposed a relevance criterion for eliminating irrelevant predicates. This criterion was based on the equivalence of their success sets. Future work is needed to investigate the relaxation of this criterion to the agreement of success sets up to some tolerance threshold. Given $\epsilon > 0$ and $\delta > 0$, two predicates are said to be ϵ,δ-equivalent if their success sets agree on a high proportion of the instance space $1 - \epsilon$ with high probability $1 - \delta$. Alternative relevance criteria can be based on the frequency of usage across tasks in a continuous learning setting or on the syntactical uniqueness of the learned definition after an unfold operation [Cropper, 2020]. Finally, we suggest prioritising invented definitions using heuristic tests based on the compression of the proofs of positive examples [Vyskočil *et al.*, 2010].

Learning Bias Chapter 4 has focused on learning in a Bayesian setting. However, we have assumed that a prior probability for each candidate hypothesis was known and provided as input, and that this distributional bias was fixed. Chapter 5 has introduced a method for bias shift which automatically builds a set of intermediate concepts to solve a task more efficiently. In Chapter 6, *MIGO* performs bias shift and identifies invented predicates useful to solve multiple related tasks more efficiently. However, neither Chapter 5 nor Chapter 6 benefit from the flexibility of the Bayesian representation. Moreover, in these chapters, distributional assumptions over the hypothesis space are implicit and fixed. Future work could investigate learning explicit distributional Bayesian bias. This problem considers a learner embedded within an environment of related tasks sampled from the same underlying probability distribution. This learner aims at automatically learning a bias, as a distribution over the hypothesis space, that is appropriate for the environment of all tasks. The learner acquires information about the environment by solving tasks sampled from the hypothesis space after observing examples sampled for each learning problem from the instance space. Bias learning is a form of learning to learn [Baxter, 1998; Baxter, 2000]. Within Bayesian MIL [Muggleton *et al.*, 2014], a probability distribution over the hypothesis space can be defined by assigning a sampling probability to each primitive predicate and each meta-rule. In this case, learning a bias involves the identification of probability parameters for each primitive predicate and meta-rule and the automatic selection of additional relevant invented predicates. Selection of relevant predicates will be based on the difference between the probabilities of sampling a definition and the frequency of usage of this definition observed in previous tasks. The idea is to choose which predicates to include in the background knowledge, and which probability to assign them, based on the difficulty of relearning them and their usage, expressed as sampling probabilities.

7.2.3 Meta-Interpretive Learning

Handling noise We have assumed in our experiments noise-free data. For instance, we have assumed in Chapter 4 the oracle makes no mistake when labelling examples. However, the observation of the outcome of real-world experiments may contain errors. Similarly, we have assumed

in Chapter 5 that the background knowledge was correct and relations it contains were true. Conversely, in most practical applications, data might be imperfect and include inaccuracies in the examples, the background knowledge or both [Nienhuys-Cheng, 1997]. Noisy examples can be identified using the minimal description length [Rissanen, 1978]: because noisy examples thwart regularities, the simplest correct theory for a given set of examples has significantly lower complexity if removing noisy examples from that set. For instance, Progol [Muggleton, 1995] deals with noisy data by using a compression measure to trade-off the description of errors against the hypothesis description length. More generally, most ILP systems based on set covering naturally support noise handling [Cropper and Dumančić, 2022]. Alternatively, a noise tolerant version of *Metagol* [Muggleton *et al.*, 2018a] finds hypotheses consistent with randomly selected subsets of the training examples, evaluates each resulting hypothesis on the remaining part of the training set, and returns the hypothesis with the highest score. The noise-tolerant version of ILASP [Law *et al.*, 2018] uses ASP's optimisation ability to provably learn the program with the best coverage. Neural approaches [Evans and Grefenstette, 2018] naturally can handle noisy input data. Future work will need to be conducted to investigate handling of noisy examples in MIL using ensemble methods or representing uncertainty about the examples with probabilistic approaches. Similarly, future work could incorporate uncertainty about the background knowledge in the form of probabilities. An interesting perspective is to extend the immediate consequence operator introduced in Chapter 5 to make use of probabilities in the learned definitions. Probabilistic Inductive Logic Programming is a framework introducing probabilistic reasoning into logic programs to represent uncertain information [De Raedt and Kersting, 2008].

Probabilistic MIL We have considered in Chapter 4 that experiments are the observation of a deterministic binary output for a particular set-up. Chapter 6 has considered fully-observable deterministic games which do not involve randomness. Future work will be needed to account for possible randomness in the experimental observations. Experiments could be represented as probabilistic observations from which the aim is to learn probabilistic rules. Probabilistic Inductive Logic Programming representations [De Raedt and Kersting, 2008] extends the back-

ground knowledge and examples with probabilities indicating the degree of confidence in its correctness. For instance, Stochastic Logic Programs [Muggleton, 1996] or Problog [De Raedt *et al.*, 2007] provide frameworks to capture uncertainty in the form of explicit probabilities. Stochastic Logic Programs define a distribution over derivations, clauses are annotated with probabilities representing the probability of being sampled from the model. Conversely, Problog programs define a distribution over logic programs by specifying for each clause the probability that it belongs to a randomly sampled program. In this case, probabilities represent degree of belief and are mutually independent. Learning probabilistic relational models involves maximising a probabilistic score, such as the likelihood of correctly covering the examples. For instance, in Bayesian MIL, a super imposed logic program can be formed by labelling each arc with the sum of the posterior probabilities of sampled hypotheses containing that arc [Muggleton *et al.*, 2013]. Such a learned model can be scored on a test set based on the sum of log posterior probabilities computed as the sum of the prior of the model and the likelihood of the observations given the model. Alternatively, ProbFOIL+ [De Raedt *et al.*, 2015] extends FOIL [Quinlan, 1990] and learns probabilistic programs from probabilistic examples and background knowledge. We suggest future work is needed to investigate MIL of probabilistic programs.

Applications The string transformation experiment described in Chapter 5 is inspired from real-world problems. However, experiments presented in Chapters 4 and 6 and the chess experiment presented in Chapter 5 take place in controlled environments. Further experiments could aim at demonstrating the scalability of our contributions over a wider range of domains associated with more realistic problems including real-world datasets. In Chapter 4, our framework has been evaluated on artificial domains. We believe our active learning framework presented is valuable for multiple applications in various AI domains in which acquiring labelled observations has some costs, for instance the modelling of robotic, animal or human behaviour. In Chapter 5, the scalability of our approach is limited by the monotonic increase of the number of predicate symbols and the complexity reliance over the number of constant symbols. Future work will be needed to investigate how an improved selection of relevant predicates and a reduced dependence on the number of constant symbols can improve scalability. Potential

future applications of our method include learning more complex game strategies. In Chapter 6, we have considered a limited and simple set of evaluable games, in which the Minimax Regret can be evaluated. Future work could aim to demonstrate the capabilities of *MIGO* over a wider range of games. The General Game Playing framework [Genesereth *et al.*, 2005; Genesereth and Thielscher, 2014] evaluates an agent's general capability to learn any arbitrary game and provides a challenging test bed. Similar to *MIGO*, General Game Players are given the rules of a game. These rules specify the initial game states, what constitutes legal moves, how moves update the game state, how the game terminates and what the outcome is. The performance is evaluated as the total score over all games. To sum up, the frameworks presented in this thesis have been demonstrated suitable for accurately identifying known target strategies, which was necessary to validate the performance. In the future, these frameworks could be used to uncover logic programs representing unknown and novel scientific knowledge. Future work could aim at inducing novel agent strategies and game strategies.

Virtual Conventional Learning We have demonstrated in Chapter 6 that the learned game strategies can provide some form of comprehensibility. Visual and verbal explanations were generated from the learned strategy and presented to human participants [Ai *et al.*, 2021]. However, human-computer communication was fixed and limited. More generally, human-computer interaction presently is rigid and requires humans to adapt their behaviour to inflexible pre-programmed communication protocols. By contrast, human-human interaction uses flexible and adaptable communication protocols developed instantaneously. Recent work in cognitive psychology describes the human ability to spontaneously develop new effective and creative communication protocols for joint problem-solving [Misyak *et al.*, 2016; Chater and Misyak, 2021]. These communication protocols are virtual conventions: they are virtually negotiated and emerge from limited non-verbal interactions, yet they are equivalent to more explicitly negotiated protocols [Misyak and Chater, 2014]. These protocols can be used with great flexibility and may be changed and reshaped during acquisition and use. Recent work shows that virtual conventions can be replicated in a logic-based representation and be adapted as revision of pre-existing conventions [Bundy *et al.*, 2021]. Future work could aim

at learning virtual conventions. Machines virtually negotiate with humans or other machines, resulting in the mutual learning of explainable virtual conventions for effective interactions between humans and computers under limited communication. Learning virtual conventions involves a cooperative game between two agents, each having incomplete knowledge of both the rules and appropriate strategies for effective play. At the end of each game sequence, the outcome of the game is provided to both players as a measure of joint success. These outcomes could be used by an extension of the system *MIGO* presented in Chapter 6 to generate measurably comprehensible conventions. These conventions could be learned as game strategies and facilitate rapid interactions between agents in bandwidth limited situations.

7.3 Summary of Thesis Achievements

We have considered in this thesis the problem of inducing hypotheses, as logic programs, from examples. ILP is a form of Machine Learning for inducing logic program hypotheses that generalise examples. MIL is a state-of-the-art ILP approach which supports predicate invention, learning of recursive programs and learning of higher-order programs. However, current MIL approaches suffer from limited efficiency: the sample complexity is polynomial in the size of learned programs and the learning complexity is exponential in the size of learned programs. We have investigated in this thesis methods to achieve more efficient MIL. Specifically, we have introduced methods that revise the instance space, methods that revise the hypothesis space and methods that revise both the instance space and the hypothesis space to achieve more efficient MIL. We have demonstrated these methods can improve the sample and learning efficiency in MIL. First, we have introduced Active Bayesian MIL which supports automated experiment selection with active learning. We have demonstrated Active Bayesian MIL can significantly reduce the sample complexity in MIL. Second, we have introduced a novel predicate invention method based upon an extension of the immediate consequence operator to second-order logic programs. This method generates a set of reusable predicates in a pre-processing step. We have demonstrated this method can reduce the sample and learning complexity in MIL. Finally, we have introduced a novel MIL system for learning optimal strategies for two-player evaluable

games. Our system learns from playing: instance selection is guided by the current learned strategy. Moreover, the learning task is divided into interrelated subtasks which are jointly learned thus changing the bias. We have demonstrated our system achieves significantly lower sample complexity and improved learning performance compared to reinforcement learning systems.

We believe the contributions of this thesis open new inspiring perspectives for learning more efficiently theories with MIL in a wide range of applications including robotics, modelling of agent strategies and game playing.

Bibliography

[Adadi and Berrada, 2018] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on Explainable Artificial Intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018.

[Adé *et al.*, 1995] Hilde Adé, Luc De Raedt, and Maurice Bruynooghe. Declarative bias for specific-to-general ILP systems. *Machine Learning*, 20(1):119–154, 1995.

[Ai *et al.*, 2021] Lun Ai, Stephen H Muggleton, Céline Hocquette, Mark Gromowski, and Ute Schmid. Beneficial and harmful explanatory machine learning. *Machine Learning*, 110(4):695–721, 2021.

[Angluin, 1987] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[Angluin, 1988] Dana Angluin. Queries and concept learning. *Journal of Automated Reasoning*, 2(4):319–42, 1988.

[Apt and Van Emden, 1982] Krzysztof R. Apt and Maarten H. Van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, 1982.

[Arias *et al.*, 2007] Marta Arias, Roni Khardon, and Jérôme Maloberti. Learning horn expressions with LOGAN-H. *Journal of Machine Learning Research*, 8(20):549–587, 2007.

[Awasthi *et al.*, 2013] Pranjal Awasthi, Vitaly Feldman, and Varun Kanade. Learning using local membership queries. In *COLT*, volume 30 of *JMLR Workshop and Conference Proceedings*, pages 398–431. JMLR.org, 2013.

[Bain and Muggleton, 1994] Michael Bain and Stephen Muggleton. Learning optimal chess strategies. In *Machine Intelligence 13: Machine Intelligence and Inductive Learning*, pages 291–309. 1994.

[Bain and Srinivasan, 2018] Michael Bain and Ashwin Srinivasan. Identification of biological transition systems using meta-interpreted logic programs. *Machine Learning*, 107(7):1171–1206, 2018.

[Balog *et al.*, 2017] Matej Balog, Alexander Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. In *International Conference on Learning Representations (ICLR 2017)*. OpenReview. net, 2017.

[Baram *et al.*, 2004] Yoram Baram, Ran El Yaniv, and Kobi Luz. Online choice of active learning algorithms. *Journal of Machine Learning Research*, 5:255–291, 2004.

[Baum and Lang, 1992] Eric B Baum and Kenneth Lang. Query learning can work poorly when a human oracle is used. In *International Joint Conference on Neural Networks*, volume 8, pages 335–340, 1992.

[Baxter, 1998] Jonathan Baxter. Theoretical models of learning to learn. In *Learning to Learn*, pages 71–94. Springer, 1998.

[Baxter, 2000] Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[Blockeel and De Raedt, 1998] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[Blockeel *et al.*, 1999] Hendrik Blockeel, Luc De Raedt, Nico Jacobs, and Bart Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3(1):59–93, 1999.

[Blumer *et al.*, 1985] Anselm Blumer, Janet Blumer, David Haussler, Andrzej Ehrenfeucht, Mu-Tian Chen, and Joel Seiferas. The smallest automation recognizing the subwords of a text. *Theoretical Computer Science*, 40:31–55, 1985.

[Blumer *et al.*, 1989] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

[Bohan *et al.*, 2011] David A. Bohan, Geoffrey Caron-Lormier, Stephen H. Muggleton, Alan Raybould, and Alireza Tamaddoni-Nezhad. Automated discovery of food webs from ecological data using logic-based machine learning. *PLoS One*, 6(12):1–9, 2011.

[Bohan *et al.*, 2017] David A. Bohan, Corinne Vacher, Alireza Tamaddoni-Nezhad, Alan Raybould, Alex J. Dumbrell, and Guy Woodward. Next-generation global biomonitoring: Large-scale, automated reconstruction of ecological networks. *Trends in Ecology & Evolution*, 32(7):477–487, 2017.

[Bratko, 1978] Ivan Bratko. Proving correctness of strategies in the AL1 assertional language. *Information Processing Letters*, 7(5):223–230, 1978.

[Bratko, 2012] Ivan Bratko. *Prolog Programming for Artificial Intelligence, 4th Edition*. Addison-Wesley, 2012.

[Bridewell and Todorovski, 2007] Will Bridewell and Ljupčo Todorovski. Learning declarative bias. In *International Conference on Inductive Logic Programming*, pages 63–77. Springer, 2007.

[Brooks, 2017] Rodney Brooks. FoR & AI: Machine learning explained, 2017. https://rodneybrooks.com/forai-machine-learning-explained/.

[Bryant *et al.*, 2001] Christopher H Bryant, SH Muggleton, DB Kell, P Reiser, RD King, SG Oliver, et al. Combining inductive logic programming, active learning and robotics to discover the function of genes. *Electronic Transactions in Artificial Intelligence*, 5(B):1–36, 2001.

[Buchanan *et al.*, 1969] Bruce Buchanan, Georgia Sutherland, and Edward A. Feigenbaum. Heuristic DENDRAL: A program for generating explanatory hypotheses. *Organic Chemistry*, 1969.

[Bundy and Mitrovic, 2016] Alan Bundy and Boris Mitrovic. Reformation: A domain-independent algorithm for theory repair. Working paper, University of Edinburgh, 2016.

[Bundy *et al.*, 2021] Alan Bundy, Eugene Philalithis, and Xue Li. Modelling virtual bargaining using logical representation change. In Stephen H. Muggleton and Nick Chater, editors, *Human-Like Machine Intelligence*, pages 68–89. Oxford university Press, Oxford, UK, 2021.

[Califf and Mooney, 2003] Mary Elaine Califf and Raymond J Mooney. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research*, 4:177–210, 2003.

[Carbonell and Gil, 1990] Jaime G. Carbonell and Yolanda Gil. Learning by experimentation: The operator refinement method. In *Machine Learning*, pages 191–213. Elsevier, 1990.

[Ceri *et al.*, 1989] Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *Knowledge and Data Engineering, IEEE Transactions on*, 1:146–166, 04 1989.

[Chater and Misyak, 2021] Nick Chater and Jennifer Misyak. Spontaneous communicative conventions through virtual bargaining. In Stephen H. Muggleton and Nick Chater, editors, *Human-Like Machine Intelligence*, pages 52–67. Oxford University Press, 2021.

[Chollet, 2019] François Chollet. On the measure of intelligence. *arXiv preprint arXiv:1911.01547*, 2019.

[Cohen, 1995] William W. Cohen. Fast effective rule induction. In *Machine Learning Proceedings 1995*, pages 115–123. Elsevier, 1995.

[Cohn *et al.*, 1994] David A. Cohn, Les E. Atlas, and Richard E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[Colmerauer and Roussel, 1996] Alain Colmerauer and Philippe Roussel. *The Birth of Prolog*, pages 331–367. Association for Computing Machinery, New York, NY, USA, 1996.

[Corapi *et al.*, 2011] Domenico Corapi, Alessandra Russo, and Emil Lupu. Inductive Logic Programming in Answer Set Programming. In Stephen H. Muggleton, Alireza Tamaddoni-Nezhad, and F. A. Lisi, editors, *Inductive Logic Programming*, pages 91–97, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Croonenborghs *et al.*, 2008] Tom Croonenborghs, Kurt Driessens, and Maurice Bruynooghe. Learning relational options for inductive transfer in relational reinforcement learning. In Hendrik Blockeel, Jan Ramon, Jude Shavlik, and Prasad Tadepalli, editors, *Inductive Logic Programming*, pages 88–97, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[Cropper and Dumančić, 2022] Andrew Cropper and Sebastijan Dumančić. Inductive logic programming at 30: a new introduction. *Journal of Artificial Intelligence Research*, 2022.

[Cropper and Morel, 2021] Andrew Cropper and Rolf Morel. Learning programs by learning from failures. *Machine Learning*, pages 1–56, 2021.

[Cropper and Muggleton, 2014] Andrew Cropper and Stephen H. Muggleton. Logical minimisation of meta-rules within meta-interpretive learning. In Jesse Davis and Jan Ramon, editors, *Inductive Logic Programming - 24th International Conference, ILP 2014, Nancy, France, 2014, Revised Selected Papers*, volume 9046 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2014.

[Cropper and Muggleton, 2015] Andrew Cropper and Stephen H. Muggleton. Learning efficient logical robot strategies involving composable objects. In *Proceedings of the 24th International Joint Conference Artificial Intelligence (IJCAI 2015)*, pages 3423–3429, 2015.

[Cropper and Muggleton, 2016] Andrew Cropper and Stephen H. Muggleton. Metagol system. https://github.com/metagol/metagol, 2016.

[Cropper and Muggleton, 2018] Andrew Cropper and Stephen H. Muggleton. Learning efficient logic programs. *Machine Learning*, 2018.

[Cropper and Tourret, 2018] Andrew Cropper and Sophie Tourret. Derivation reduction of metarules in meta-interpretive learning. In Fabrizio Riguzzi, Elena Bellodi, and Riccardo

Zese, editors, *Inductive Logic Programming - 28th International Conference, ILP 2018, Ferrara, Italy, Proceedings*, volume 11105 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2018.

[Cropper and Tourret, 2020] Andrew Cropper and Sophie Tourret. Logical reduction of metarules. *Machine Learning*, 109(7):1323–1369, 2020.

[Cropper *et al.*, 2020a] Andrew Cropper, Sebastijan Dumančić, and Stephen H. Muggleton. Turning 30: New ideas in Inductive Logic Programming. In *Proceedings of the Twenty-Nineth International Joint Conference on Artificial Intelligence, IJCAI*, pages 4833–4839, 2020.

[Cropper *et al.*, 2020b] Andrew Cropper, Rolf Morel, and Stephen Muggleton. Learning higher-order logic programs. *Machine Learning*, 109(7):1289–1322, 2020.

[Cropper, 2019] Andrew Cropper. Playgol: Learning Programs Through Play. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, 2019*, pages 6074–6080. ijcai.org, 2019.

[Cropper, 2020] Andrew Cropper. Forgetting to learn logic programs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3676–3683, 2020.

[Culotta and McCallum, 2005] Aron Culotta and Andrew McCallum. Reducing labeling effort for structured prediction tasks. In *Proceedings of the 20th National Conference on Artificial intelligence*, volume 5, pages 746–751, 2005.

[Dai *et al.*, 2017] Wang-Zhou Dai, Stephen H. Muggleton, Jing Wen, Alireza Tamaddoni-Nezhad, and Zhi-Hua Zhou. Logical vision: One-shot meta-interpretive learning from real images. In Nicolas Lachiche and Christel Vrain, editors, *Inductive Logic Programming - 27th International Conference, ILP 2017, Orléans, France, Revised Selected Papers*, volume 10759 of *Lecture Notes in Computer Science*, pages 46–62. Springer, 2017.

[Dasgupta and Hsu, 2008] Sanjoy Dasgupta and Daniel Hsu. Hierarchical sampling for active learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 208–215, 2008.

[Dasgupta, 2005a] Sanjoy Dasgupta. Analysis of a greedy active learning strategy. *Advances in Neural Information Processing Systems*, 17:337–344, 2005.

[Dasgupta, 2005b] Sanjoy Dasgupta. Coarse sample complexity bounds for active learning. *Proceedings of the 18th International Conference on Neural Information Processing Systems*, pages 235,242, 2005.

[De Raedt and Bruynooghe, 1992] Luc De Raedt and Maurice Bruynooghe. Interactive concept-learning and constructive induction by analogy. *Machine Learning*, 8(2):107–150, 1992.

[De Raedt and Džeroski, 1994] Luc De Raedt and Sašo Džeroski. First-order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70(1-2):375–392, 1994.

[De Raedt and Kersting, 2008] Luc De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In *Probabilistic Inductive Logic Programming*, pages 1–27. Springer, 2008.

[De Raedt *et al.*, 2007] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 2468–2473, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[De Raedt *et al.*, 2015] Luc De Raedt, Anton Dries, Ingo Thon, Guy Van Den Broeck, and Mathias Verbeke. Inducing probabilistic relational rules from probabilistic examples. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 1835–1843, 2015.

[De Raedt, 1997] Luc De Raedt. Logical settings for concept-learning. *Artificial Intelligence*, 95(1):187–201, 1997.

[De Raedt, 2012] Luc De Raedt. Declarative modeling for machine learning and data mining. In Florent Domenach, Dmitry I. Ignatov, and Jonas Poelmans, editors, *Formal Concept Analysis*, pages 2–2, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[Domingos, 1999] Pedro Domingos. The role of occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3(4):409–425, 1999.

[Driessens and Džeroski, 2004] Kurt Driessens and Sašo Džeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, 2004.

[Driessens and Ramon, 2003] Kurt Driessens and Jan Ramon. Relational instance based regression for relational reinforcement learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 123–130, 2003.

[Driessens *et al.*, 2001] Kurt Driessens, Jan Ramon, and Hendrik Blockeel. Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In *European Conference on Machine Learning*, pages 97–108. Springer, 2001.

[Dumancic *et al.*, 2021] Sebastijan Dumancic, Tias Guns, and Andrew Cropper. Knowledge refactoring for inductive program synthesis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7271–7278, 2021.

[Dumančić and Blockeel, 2017] Sebastijan Dumančić and Hendrik Blockeel. Clustering-based relational unsupervised representation learning with an explicit distributed representation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 1631–1637. AAAI Press, 2017.

[Dumančić *et al.*, 2019] Sebastijan Dumančić, Tias Guns, Wannes Meert, and Hendrik Blockeel. Learning relational representations with auto-encoding logic programs. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, 2019*, pages 6081–6087. ijcai.org, 2019.

[Džeroski *et al.*, 2001] Sašo Džeroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43(1):7–52, 2001.

[Ebert *et al.*, 2012] Sandra Ebert, Mario Fritz, and Bernt Schiele. Ralf: A reinforced active learning formulation for object class recognition. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3626–3633. IEEE, 2012.

[Ellis *et al.*, 2018] Kevin Ellis, Lucas Morales, Mathias Sablé-Meyer, Armando Solar-Lezama, and Josh Tenenbaum. Learning libraries of subroutines for neurally–guided bayesian program induction. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 7805–7815. Curran Associates, Inc., 2018.

[Emde *et al.*, 1983] Werner Emde, Christopher U. Habel, and Claus-Rainer Rollinger. The discovery of the equator or concept driven learning. In *Proceedings of the Eighth International Joint Conference on Artificial intelligence-Volume 1*, pages 455–458, 1983.

[Evans and Grefenstette, 2018] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

[Evans *et al.*, 2021] Richard Evans, José Hernández-Orallo, Johannes Welbl, Pushmeet Kohli, and Marek Sergot. Making sense of sensory input. *Artificial Intelligence*, 293:103438, 2021.

[Fensel and Wiese, 1993] Dieter Fensel and Markus Wiese. Refinement of rule sets with jojo. In Pavel B. Brazdil, editor, *Machine Learning: ECML-93*, pages 378–383, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[fheisler, 2015] fheisler. Q-learning tic-tac-toe. `https://gist.github.com/fheisler/430e70fa249ba30e707f`, 2015.

[Flach, 1993] Peter A Flach. Predicate invention in inductive data engineering. In *European Conference on Machine Learning*, pages 83–94. Springer, 1993.

[Flener and Yıilmaz, 1999] Pierre Flener and Serap Yıilmaz. Inductive synthesis of recursive logic programs: Achievements and prospects. *The Journal of Logic Programming*, 41(2-3):141–195, 1999.

[Flener, 1996] Pierre Flener. Inductive logic program synthesis with dialogs. In *International Conference on Inductive Logic Programming*, pages 175–198. Springer, 1996.

[Freund *et al.*, 1997] Yoav Freund, H. Sebastian Seung, Eli Shamir, and Naftali Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, pages 1551–1557, 1997.

[Gärdenfors and Rott, 1995] Peter Gärdenfors and Hans Rott. Belief revision. *Handbook of Logic in Artificial Intelligence and Logic Programming*, 4:35–132, 1995.

[Gardner, 1962] Martin Gardner. Mathematical games, scientific american. *reprinted in The Unexpected Hanging and Other Mathematical Diversions*, page 93, 1962.

[Garnelo *et al.*, 2016] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. *arXiv preprint arXiv:1609.05518*, 2016.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. *Proceedings of International Logic Programming Conference and Symposium*, 2(1):1070–1080, 1988.

[Genesereth and Thielscher, 2014] Michael Genesereth and Michael Thielscher. General game playing. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(2):1–229, 2014.

[Genesereth *et al.*, 2005] Michael Genesereth, Nathaniel Love, and Barney Pell. General game playing: Overview of the AAAI competition. *AI magazine*, 26(2):62–62, 2005.

[Gil *et al.*, 2014] Yolanda Gil, Mark Greaves, James Hendler, and Haym Hirsh. Amplify scientific discovery with artificial intelligence. *Science*, 346(6206):171–172, 2014.

[Gold, 1967] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[Gulwani *et al.*, 2015] Sumit Gulwani, José Hernández-Orallo, Emanuel Kitzelmann, Stephen H Muggleton, Ute Schmid, and Benjamin Zorn. Inductive programming meets the real world. *Communications of the ACM*, 58(11):90–99, 2015.

[Gulwani, 2011] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium*

on Principles of Programming Languages, POPL '11, pages 317–330, New York, NY, USA, 2011. ACM.

[Gunning and Aha, 2019] David Gunning and David Aha. DARPA's explainable artificial intelligence (XAI) program. *AI Magazine*, 40(2):44–58, 2019.

[Hanneke, 2007] Steve Hanneke. A bound on the label complexity of agnostic active learning. In *Proceedings of the 24th International Conference on Machine Learning*, pages 353–360, 2007.

[Hanneke, 2014] Steve Hanneke. Theory of disagreement-based active learning. *Foundations and Trends® in Machine Learning*, 7(2-3):131–309, 2014.

[Haussler *et al.*, 1994] David Haussler, Michael Kearns, and Robert E. Schapire. Bounds on the sample complexity of bayesian learning using information theory and the VC-dimension. *Machine Learning*, 14:83–113, 1994.

[Hino, 2020] Hideitsu Hino. Active learning: Problem settings and recent developments. *arXiv preprint arXiv:2012.04225*, 2020.

[Hocquette and Muggleton, 2018] Céline Hocquette and Stephen H. Muggleton. How much can experimental cost be reduced in active learning of agent strategies? In Fabrizio Riguzzi, Elena Bellodi, and Riccardo Zese, editors, *Inductive Logic Programming*, pages 38–53, Cham, 2018. Springer International Publishing.

[Hocquette and Muggleton, 2020] Céline Hocquette and Stephen H Muggleton. Complete bottom-up predicate invention in meta-interpretive learning. In *Proceedings of the 29th International Joint Conference Artificial Intelligence*, pages 2312–2318, 2020.

[Hsu and Lin, 2015] Wei-Ning Hsu and Hsuan-Tien Lin. Active learning by learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2659–2665, 2015.

[Inoue *et al.*, 2009] Katsumi Inoue, Koichi Furukawa, Ikuo Kobayashi, and Hidetomo Nabeshima. Discovering rules by meta-level abduction. In *International Conference on Inductive Logic Programming*, pages 49–64. Springer, 2009.

[Kaminski *et al.*, 2018] Tobias Kaminski, Thomas Eiter, and Katsumi Inoue. Exploiting answer set programming with external sources for meta-interpretive learning. *Theory and Practice of Logic Programming*, 18(3-4):571–588, 2018.

[Kersting *et al.*, 2004] Kristian Kersting, Martijn Van Otterlo, and Luc De Raedt. Bellman goes relational. In *Proceedings of the Twenty-First International Conference on Machine Learning*, page 59, 2004.

[Kietz and Wrobel, 1992] Jörg-Uwe Kietz and Stefan Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In *Inductive Logic Programming*, pages 335–359. Academic Press, 1992.

[Kijsirikul *et al.*, 1992] Boonserm Kijsirikul, Masayuki Numao, and Masamichi Shimura. Discrimination-based constructive induction of logic programs. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 44–49, 1992.

[King *et al.*, 2004] Ross D. King, Kenneth E. Whelan, Ffion M. Jones, Philip K.G. Reiser, Christopher H. Bryant, Stephen H. Muggleton, Douglas B. Kell, and Stephen G. Oliver. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature 427*, pages 247–252, 2004.

[King *et al.*, 2009] Ross D King, Jem Rowland, Stephen G Oliver, Michael Young, Wayne Aubrey, Emma Byrne, Maria Liakata, Magdalena Markham, Pinar Pir, Larisa N Soldatova, et al. The automation of science. *Science*, 324(5923):85–89, 2009.

[Kok and Domingos, 2007] Stanley Kok and Pedro Domingos. Statistical predicate invention. In *Proceedings of the Twenty-Fourth International Conference on Machine Learning*, pages 433–440, 2007.

[Kokel *et al.*, 2021] Harsha Kokel, Arjun Manoharan, Sriraam Natarajan, Balaraman Ravindran, and Prasad Tadepalli. Reprel: Integrating relational planning and reinforcement learn-

ing for effective abstraction. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, pages 533–541, 2021.

[Konyushkova *et al.*, 2017] Ksenia Konyushkova, Sznitman Raphael, and Pascal Fua. Learning active learning from data. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4228–4238, 2017.

[Kowalski and Kuehner, 1971] Robert A. Kowalski and Donald Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2(3/4):227–260, 1971.

[Kowalski, 1979] Robert A. Kowalski. *Logic for Problem Solving*, volume 7 of *The Computer Science Library : Artificial Intelligence Series*. North-Holland, 1979.

[Kramer, 1995] Stefan Kramer. Predicate invention: A comprehensive view. Technical report, Austrian Research Institute for Artificial Intelligence, Vienna, Austria, 1995.

[Kramer, 2020] Stefan Kramer. A brief history of learning symbolic higher-level representations from data (and a curious look forward). In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 4868–4876. International Joint Conferences on Artificial Intelligence Organization, 2020. Survey track.

[Kulkarni *et al.*, 1993] Sanjeev R. Kulkarni, Sanjoy K. Mitter, and John N. Tsitsiklis. Active learning using arbitrary binary valued queries. *Machine Learning*, 11:23–35, 1993.

[Kurzweil, 2013] Ray Kurzweil. *How to create a mind: the secret of human thought revealed.* Penguin, 2013.

[Lake *et al.*, 2017] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

[Lang *et al.*, 2010] Tobias Lang, Marc Toussaint, and Kristian Kersting. Exploration in relational worlds. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 178–194. Springer, 2010.

[Langley *et al.*, 1987] Pat Langley, Herbert A Simon, Gary L Bradshaw, and Jan M Zytkow. *Scientific Discovery: Computational Explorations of the Creative Processes*. MIT press, 1987.

[Law *et al.*, 2018] Mark Law, Alessandra Russo, and Krysia Broda. Inductive learning of answer set programs from noisy examples. *arXiv preprint arXiv:1808.08441*, 2018.

[Law *et al.*, 2019] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. Representing and learning grammars in answer set programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2919–2928, 2019.

[Law *et al.*, 2020] Mark Law, Alessandra Russo, Elisa Bertino, Krysia Broda, and Jorge Lobo. Fastlas: scalable inductive logic programming incorporating domain-specific optimisation criteria. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2877–2885, 2020.

[Law *et al.*, 2021] Mark Law, Alessandra Russo, Krysia Broda, and Elisa Bertino. Scalable non-observational predicate learning in ASP. In Zhi-Hua Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pages 1936–1943. ijcai.org, 2021.

[Law, 2018] Mark Law. *Inductive Learning of Answer Set Programs*. PhD thesis, University of London, 2018.

[Legras *et al.*, 2018] Swann Legras, Céline Rouveirol, and Véronique Ventos. The game of bridge: A challenge for ILP. In Fabrizio Riguzzi, Elena Bellodi, and Riccardo Zese, editors, *Inductive Logic Programming*, pages 72–87, Cham, 2018. Springer International Publishing.

[Lewis and Gale, 1994] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12. ACM/Springer, 1994.

[Li *et al.*, 2018] Xue Li, Alan Bundy, and Alan Smaill. ABC repair system for datalog-like theories. In *10th International Conference on Knowledge Engineering and Ontology Development*, pages 335–342, 2018.

[Lin *et al.*, 2014] Dianhuan Lin, Eyal Dechter, Kevin Ellis, Joshua B. Tenenbaum, and Stephen H. Muggleton. Bias reformulation for one-shot function induction. *In Proceedings of the 23rd European Conference on Artificial Intelligence (ECAI 2014)*, pages 525–530, 2014.

[Lloyd, 1987] John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987.

[Maher, 1988] Michael J Maher. Equivalences of logic programs. In *Foundations of Deductive Databases and Logic Programming*, pages 627–658. Elsevier, 1988.

[Marcus and Davis, 2019] Gary Marcus and Ernest Davis. *Rebooting AI: Building artificial intelligence we can trust*. Vintage, 2019.

[Marcus, 2018] Gary Marcus. Deep learning: A critical appraisal. *arXiv preprint arXiv:1801.00631*, 2018.

[McCreath and Sharma, 1995] Eric McCreath and Arun Sharma. Extraction of meta-knowledge to restrict the hypothesis space for ILP systems. In *Eighth Australian Joint Conference on Artificial Intelligence*, pages 75–82, 1995.

[Michie, 1963] Donald Michie. Experiments on the mechanization of game-learning part i. characterization of the model and its parameters. *The Computer Journal, Volume 6, Issue 3*, pages 232–236, 1963.

[Michie, 1983] Donald Michie. Inductive rule generation in the context of the fifth generation. *Machine Learning Workshop*, pages 65–70, 1983.

[Michie, 1988] Donald Michie. Machine learning in the next five years. In *Proceedings of the Third European Working Session on Learning*, pages 107–122. Pitman, 1988.

[Miller, 1956] George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97, 1956.

[Minsky, 1961] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 1(49):8–30, 1961.

[Misyak and Chater, 2014] Jennifer B. Misyak and Nick Chater. Virtual bargaining: A theory of social decision-making. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 369(1655):20130487, 2014.

[Misyak *et al.*, 2016] Jennifer Misyak, Takao Noguchi, and Nick Chater. Instantaneous conventions: The emergence of flexible communicative signals. *Psychological Science*, 27(12):1550–1561, 2016.

[Mitchell *et al.*, 2018] Tom Mitchell, William Cohen, Estevam Hruschka, Partha Talukdar, Joel Yang, Justin Betteridge, Andrew Carlson, Bhavana Dalvi, Matthew Gardner, Bryan Kisiel, Jayant Krishnamurthy, Ni Lao, Kathryn Mazaitis, Thahir Mohamed, Ndapa Nakashole, Emmanouil Platanios, Alan Ritter, Mehdi Samadi, Burr Settles, Richard Wang, Derry Wijaya, Abhinav Gupta, Xinlei Chen, Abulhair Saparov, Malcolm Greaves, and Joel Welling. Never-ending learning. *Communications of the ACM*, 61(5):103–115, 2018.

[Mitchell, 1977] Tom M. Mitchell. Version spaces: A candidate elimination approach to rule learning. In Raj Reddy, editor, *Proceedings of the 5th International Joint Conference on Artificial Intelligence. Cambridge, MA, USA, 1977*, pages 305–310. William Kaufmann, 1977.

[Mitchell, 1978] Tom M. Mitchell. Version spaces: An approach to concept learning. *PhD Thesis*, 1978.

[Mitchell, 1980] Tom M. Mitchell. The need for biases in learning generalizations. pages 184–191. Department of Computer Science, Laboratory for Computer Science Research, 1980.

[Mitchell, 1982] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[Mitchell, 1997] Tom M. Mitchell. Machine learning. *McGraw Hill*, 1997.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[Moyle and Muggleton, 1997] Stephen Moyle and Stephen H. Muggleton. Learning programs in the event calculus. In *International Conference on Inductive Logic Programming*, pages 205–212. Springer, 1997.

[Muggleton and Bryant, 2000] Stephen H. Muggleton and Christopher H. Bryant. Theory completion using inverse entailment. In *International Conference on Inductive Logic Programming*, pages 130–146. Springer, 2000.

[Muggleton and Buntine, 1988] Stephen H. Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In John Laird, editor, *Machine Learning Proceedings 1988*, pages 339–352. Morgan Kaufmann, San Francisco (CA), 1988.

[Muggleton and De Raedt, 1994] Stephen H. Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19-20:629–679, 1994. Special Issue: Ten Years of Logic Programming.

[Muggleton and Feng, 1990] Stephen H. Muggleton and Cao Feng. Efficient induction of logic programs. In Setsuo Arikawa, Shigeki Goto, Setsuo Ohsuga, and Takashi Yokomori, editors, *Algorithmic Learning Theory, First International Workshop, ALT '90, Tokyo, Japan, 1990, Proceedings*, pages 368–381. Springer/Ohmsha, 1990.

[Muggleton and Hocquette, 2019] Stephen H. Muggleton and Céline Hocquette. Machine discovery of comprehensible strategies for simple games using meta-interpretive learning. *New Generation Computing*, 37(2):203–217, 2019.

[Muggleton and Page, 1994] Stephen Muggleton and CD Page. A learnability model for universal representations. In *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237, pages 139–160. Citeseer, 1994.

[Muggleton and Tamaddoni-Nezhad, 2008] Stephen Muggleton and Alireza Tamaddoni-Nezhad. Qg/ga: a stochastic search for progol. *Machine Learning*, 70(2):121–133, 2008.

[Muggleton *et al.*, 2012] Stephen H. Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter A. Flach, Katsumi Inoue, and Ashwin Srinivasan. ILP turns 20 - biography and future challenges. *Machine Learning*, 86(1):3–23, 2012.

[Muggleton *et al.*, 2013] Stephen H Muggleton, Dianhuan Lin, Jianzhong Chen, and Alireza Tamaddoni-Nezhad. Metabayes: Bayesian meta-interpretative learning using higher-order stochastic refinement. In *International Conference on Inductive Logic Programming*, pages 1–17. Springer, 2013.

[Muggleton *et al.*, 2014] Stephen H. Muggleton, Dianhuan Lin, Niels Pahlavi, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning: application to grammatical inference. *Machine Learning*, 94:25–49, 2014.

[Muggleton *et al.*, 2015] Stephen H. Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Machine Learning*, 100(1):49–73, 2015.

[Muggleton *et al.*, 2018a] Stephen H. Muggleton, Wang-Zhou Dai, Claude Sammut, Alireza Tamaddoni-Nezhad, Jing Wen, and Zhi-Hua Zhou. Meta-interpretive learning from noisy images. *Machine Learning*, 107(7):1097–1118, 2018.

[Muggleton *et al.*, 2018b] Stephen H. Muggleton, Ute Schmid, Christina Zeller, Alireza Tamaddoni-Nezhad, and Tarek Besold. Ultra-strong machine learning: comprehensibility of programs learned with ILP. *Machine Learning*, 107(7):1119–1140, 2018.

[Muggleton, 1987] Stephen H. Muggleton. Duce, an oracle-based approach to constructive induction. In John P. McDermott, editor, *Proceedings of the 10th International Joint Conference on Artificial Intelligence. Milan, Italy, 1987*, pages 287–292. Morgan Kaufmann, 1987.

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

[Muggleton, 1995] Stephen H. Muggleton. Inverse Entailment and Progol. *New Generation Comput.*, 13(3&4):245–286, 1995.

[Muggleton, 1996] Stephen H. Muggleton. Learning from positive data. *In S.H. Muggleton, editor, Proceedings of the Sixth International Workshop on Inductive Logic Programming (Workshop-96), LNAI 1314, Springer-Verlag*, pages 358–376, 1996.

[Muggleton, 2014] Stephen H. Muggleton. Alan Turing and the development of Artificial Intelligence. *AI Communications*, 27(1):3–10, 2014.

[Nguyen *et al.*, 2021] Hien D Nguyen, Chiaki Sakama, Taisuke Sato, and Katsumi Inoue. An efficient reasoning method on logic programming using partial evaluation in vector spaces. *Journal of Logic and Computation*, 31(5):1298–131, 2021.

[Nienhuys-Cheng, 1997] Shan-Hwei Nienhuys-Cheng. *Foundations of Inductive Logic Programming*. Lecture Notes in Artificial Intelligence ; 1228. 1st ed. 1997. edition, 1997.

[Ong *et al.*, 2005] Irene M. Ong, Inês de Castro Dutra, David Page, and Vítor Santos Costa. Mode directed path finding. In *European Conference on Machine Learning*, pages 673–681. Springer, 2005.

[OpenAI *et al.*, 2019] OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. 2019.

[Pagallo and Haussler, 1990] Giulia Pagallo and David Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 1(5):71–99, 1990.

[Passerini *et al.*, 2006] Andrea Passerini, Paolo Frasconi, Luc De Raedt, Roland Olsson, and Ute Schmid. Kernels on prolog proof trees: Statistical learning in the ILP setting. *Journal of Machine Learning Research*, 7(2):307–342, 2006.

[Patsantzis and Muggleton, 2018] Stassa Patsantzis and Stephen H. Muggleton. Which background knowledge is relevant? *In Late Breaking Paper Proceedings of the 28th International Conference on Inductive Logic Programming. CEUR*, 2018.

[Plotkin, 1969] Gordon D. Plotkin. A note on inductive generalisation. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, pages 153–163. Edinburgh University Press, Edinburgh, 1969.

[Quinlan, 1983] J. Ross Quinlan. Learning efficient classification procedures and their application to chess end games. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 463–482, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg.

[Quinlan, 1987] J. Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.

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

[Rabinovich *et al.*, 1996] Savely Rabinovich, Gregory Berkolaiko, and Shlomo Havlin. Solving nonlinear recursions. *Journal of Mathematical Physics*, 37:5828–5836, 1996.

[Ray, 2009] Oliver Ray. Nonmonotonic abductive inductive learning. *Journal of Applied Logic*, 7(3):329–340, 2009. Special Issue: Abduction and Induction in Artificial Intelligence.

[Rendell, 1985] Larry Rendell. Substantial constructive induction using layered information compression: tractable feature formation in search. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, volume 1, pages 650–658, 1985.

[Richards and Mooney, 1992] Bradley L. Richards and Raymond J. Mooney. Learning relations by pathfinding. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 50–55. AAAI Press, 1992.

[Rissanen, 1978] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[Robinson, 1965] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, 1965.

[Rodrigues *et al.*, 2011] Christophe Rodrigues, Pierre Gérard, Céline Rouveirol, and Henry Soldano. Active learning of relational action models. In *International Conference on Inductive Logic Programming*, pages 302–316. Springer, 2011.

[Romstad *et al.*, 2017] Tord Romstad, Marco Costalba, Joona Kiiski, and G Linscott. Stockfish: A strong open source chess engine. *Open Source available at https://stockfishchess.org/. Retrieved November 29th*, 2017.

[Sakakibara, 1990] Yasubumi Sakakibara. Learning context-free grammars from structural data in polynomial time. *Theoretical Computer Science*, 76(2-3):223–242, 1990.

[Sakama *et al.*, 2017] Chiaki Sakama, Katsumi Inoue, and Taisuke Sato. Linear algebraic characterization of logic programs. In *International Conference on Knowledge Science, Engineering and Management*, pages 520–533. Springer, 2017.

[Samuel, 1959] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[Sarjant *et al.*, 2011] Samuel Sarjant, Bernhard Pfahringer, Kurt Driessens, and Tony Smith. Using the online cross-entropy method to learn relational policies for playing different games. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, pages 182–189, 2011.

[Sato *et al.*, 2015] Yuichiro Sato, Hiroyuki Iida, and H. J. van den Herik. Transfer learning by inductive logic programming. In Aske Plaat, Jaap Van Den Herik, and Walter Kosters, editors, *Advances in Computer Games*, pages 223–234, Cham, 2015. Springer International Publishing.

[Scheffer *et al.*, 2001] Tobias Scheffer, Christian Decomain, and Stefan Wrobel. Active hidden markov models for information extraction. In Frank Hoffmann, David J. Hand, Niall Adams, Douglas Fisher, and Gabriela Guimaraes, editors, *Advances in Intelligent Data Analysis*, pages 309–318, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[Schmid *et al.*, 2017] Ute Schmid, Christina Zeller, Tarek Besold, Alireza Tamaddoni-Nezhad, and Stephen Muggleton. How does predicate invention affect human comprehensibility? In James Cussens and Alessandra Russo, editors, *Inductive Logic Programming*, pages 52–67, Cham, 2017. Springer International Publishing.

[Schmid, 2021] Ute Schmid. Interactive learning with mutual explanations in relational domains. In Stephen H. Muggleton and Nick Chater, editors, *Human-Like Machine Intelligence*, pages 338–354. Oxford University Press, 2021.

[Settles *et al.*, 2008] Burr Settles, Mark Craven, and Lewis Friedland. Active learning with real annotation costs. In *Proceedings of the NIPS Workshop on Cost-Sensitive Learning*, volume 1, 2008.

[Settles, 2009] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[Shannon, 1948] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[Shannon, 1950] Claude E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41(4):256–275, 1950.

[Shapiro and Niblett, 1982] Alen Shapiro and Tim Niblett. Automatic induction of classification rules for a chess endgame. In M.R.B. Clarke, editor, *Advances in Computer Chess*, volume 3, pages 73–91. Pergammon, Oxford, 1982.

[Shapiro, 1991] Ehud Y. Shapiro. Inductive inference of theories from facts. In J-L. Lassez and G.D. Plotkin, editors, *Computational logic: essays in honor of Alan Robinson*, pages 199–254. The MIT Press, 1991.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Silver *et al.*, 2017] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore

Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[Silver *et al.*, 2018] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[Sivaraman *et al.*, 2019] Aishwarya Sivaraman, Tianyi Zhang, Guy Van den Broeck, and Miryung Kim. Active inductive logic programming for code search. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 292–303, 2019.

[Srinivasan *et al.*, 2003] Ashwin Srinivasan, Ross D. King, and Michael Bain. An empirical study of the use of relevance information in inductive logic programming. *Journal of Machine Learning Research*, 4:369–383, 2003.

[Srinivasan, 2001] Ashwin Srinivasan. The ALEPH manual. *Machine Learning at the Computing Laboratory*, 2001.

[Stahl, 1993] Irene Stahl. Predicate invention in ILP - an overview. In P. B. Brazdil, editor, *Machine Learning: ECML-93*, pages 311–322, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[Stahl, 1994] Irene Stahl. On the utility of predicate invention in inductive logic programming. In F. Bergadano and L. De Raedt, editors, *Machine Learning: ECML-94*, pages 272–286, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[Stahl, 1995] Irene Stahl. The appropriateness of predicate invention as bias shift operation in ILP. *Machine Learning*, 20(1):95–117, 1995.

[Sterling and Shapiro, 1994] Leon Sterling and Ehud Shapiro. *The Art of Prolog - Advanced Programming Techniques, 2nd Ed.* MIT Press, 1994.

[Sutton and Barto, 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning , second edition: An introduction.* Adaptive computation and Machine Mearning. MIT Press, 2018.

[Sutton, 1992] Richard S. Sutton. Adapting bias by gradient descent: an incremental version of delta-bar-delta. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 171–176, 1992.

[Tadepalli *et al.*, 2004] Prasad Tadepalli, Robert Givan, and Kurt Driessens. Relational reinforcement learning: An overview. In *Proceedings of the ICML-2004 Workshop on Relational Reinforcement Learning*, pages 1–9, 2004.

[Tamaddoni-Nezhad and Muggleton, 2011] Alireza Tamaddoni-Nezhad and Stephen H. Muggleton. Stochastic refinement. *Paolo Frasconi and Francesca A. Lisi, editors, Proceedings of the 20th International Conference on Inductive Logic Programming (ILP 2010), LNAI 6489, Berlin, Springer-Verlag*, pages 222–237, 2011.

[Tamaddoni-Nezhad *et al.*, 2006] Alireza Tamaddoni-Nezhad, Raphael Chaleil, Antonis Kakas, and Stephen H. Muggleton. Application of abductive ILP to learning metabolic network inhibition from temporal data. *Machine Learning*, 64(1):209–230, 2006.

[Tärnlund, 1977] Sten-Åke Tärnlund. Horn clause computability. *BIT Numerical Mathematics*, 17(2):215–226, 1977.

[Taylor and Stone, 2007] Matthew E. Taylor and Peter Stone. Cross-domain transfer for reinforcement learning. In *Proceedings of the 24th International Conference on Machine Learning*, pages 879–886, 2007.

[Taylor and Stone, 2009] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.

[Thompson *et al.*, 1999] Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney. Active learning for natural language parsing and information extraction. In *Proceedings of the*

Sixteenth International Conference on Machine Learning, ICML '99, pages 406–414. Morgan Kaufmann Publishers Inc., 1999.

[Thrun and Mitchell, 1995] Sebastian Thrun and Tom M. Mitchell. Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1):25–46, 1995. The Biology and Technology of Intelligent Autonomous Agents.

[Torrey *et al.*, 2006] Lisa Torrey, Jude Shavlik, Trevor Walker, and Richard Maclin. Skill acquisition via transfer learning and advice taking. In *European Conference on Machine Learning*, pages 425–436. Springer, 2006.

[Torrey *et al.*, 2007] Lisa Torrey, Jude Shavlik, Trevor Walker, and Richard Maclin. Relational macros for transfer in reinforcement learning. In *International Conference on Inductive Logic Programming*, pages 254–268. Springer, 2007.

[Tosh and Dasgupta, 2017] Christopher Tosh and Sanjoy Dasgupta. Diameter-based active learning. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3444–3452. PMLR, 2017.

[Turing, 1950] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[Urbonas *et al.*, 2020] Marius Urbonas, Alan Bundy, Juan Casanova, and Xue Li. The use of max-sat for optimal choice of automated theory repairs. In Max Bramer and Richard Ellis, editors, *Artificial Intelligence XXXVII*, pages 49–63, Cham, 2020. Springer International Publishing.

[Valdés-Pérez, 1999] Raúl E. Valdés-Pérez. Principles of human–computer collaboration for knowledge discovery in science. *Artificial Intelligence*, 107(2):335–346, 1999.

[Valiant, 1984] Leslie G. Valiant. A theory of the learnable. *Artificial Intelligence and Language Processing*, pages 1134–1142, 1984.

[Van Emden and Kowalski, 1976] Maarten H. Van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[Vijayanarasimhan and Grauman, 2009] Sudheendra Vijayanarasimhan and Kristen Grauman. What's it going to cost you?: Predicting effort vs. informativeness for multi-label image annotations. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2262–2269, 2009.

[Vinyals *et al.*, 2019] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[Von Frisch, 1967] Karl Von Frisch. *The Dance Language and Orientation of Bees*. 1967.

[Vyskočil *et al.*, 2010] Jiří Vyskočil, David Stanovskỳ, and Josef Urban. Automated proof compression by invention of new definitions. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 447–462. Springer, 2010.

[Watkins and Dayan, 1992] Christopher Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, 1992.

[Watkins, 1989] Christopher Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, 1989.

[Xu and Fekri, 2021] Duo Xu and Faramarz Fekri. Interpretable model-based hierarchical reinforcement learning using inductive logic programming. *arXiv preprint arXiv:2106.11417*, 2021.

[yanji84, 2016] yanji84. Solving tic-tac-toe using deep reinforcement learning. `https://github.com/yanji84/tic-tac-toe-rl`, 2016.

[Zahavy *et al.*, 2016] Tom Zahavy, Nir Ben-Zrihem, and Shie Mannor. Graying the black box: Understanding dqns. In *International Conference on Machine Learning*, pages 1899–1908. PMLR, 2016.

[Zelle *et al.*, 1994] John M. Zelle, Raymond J. Mooney, and Joshua B. Konvisser. Combining top-down and bottom-up techniques in inductive logic programming. In William W. Cohen and Haym Hirsh, editors, *Machine Learning Proceedings 1994*, pages 343–351. Morgan Kaufmann, San Francisco (CA), 1994.