Inductive Logic Programming

Area Presentation & Future Research Directions

Merkouris Papamichail

Presented for the fulfillment of the

General Postgraduate Examination

Computer Science Department
University of Crete
Greece
Fall 2023

Abstract

In this paper we review elementary concepts from the literature of *Inductive Logic Programming* (ILP). ILP lays at the intersection of two scientific areas, machine learning and logic programming. Contrary to traditional sub-symbolic learning methods, an ILP system learns logic programmes in the form of first-order Horn clauses. We begin by reviewing poset traversal problems within structured hypothesis spaces. Then, we shift our attention to foundational properties of classical logical formalisms, with particular emphasis to clausal logic, where the allowed formulas constitute first-order Horn clauses. The latter subset of first-order logic constitutes a Turing complete representation formalism, used in the logic programming language Prolog. We examine the primary inference rule in logic programming, namely *resolution*. Subsequently, we discuss inductive operators, such as *θ-subsumption*, *least general generalisation*, *inverse resolution*, `V−`, `W−` operators and the *abduction* operator for theory revision. We also discuss representative inductive systems, such as the FOIL, HORN algorithms, and the Model Inference System, focusing on a *query-based* approach. Our discussion also encompass meta-level programming and recent developments in neurosymbolic approaches. The goal of this paper is to acquaint the reader about ILP, providing insights into its intricacies and advocating for future research directions. We believe that the reasoning frameworks developed within the realm of ILP hold promise for addressing core issues in contemporary machine learning, including *data-efficiency* and *explainability*.

Contents

Prologue

In this paper we present some results from the literature of *inductive logic programming*. Inductive logic programming (ILP) lays at the intersection between machine learning (ML) and logic programming. Instead of learning geometrical objects, like the most popular ML methods, ILP learns a *logic programme*. Additionally to the practical applications of ILP, its theoretical foundations are of particular interest since it attempts to provide a formal theory for *inductive reasoning*. In essence, it aims to establish a formalisation for automated scientific reasoning.

The scientific method proceeds with some elementary, usually discrete and recognisable steps.

***Observation*,** we experience a natural phenomenon occurring.

***Hypothesis formulation*,** based on our background knowledge we form a hypothesis in order to explain the phenomenon.

***Experimentation*,** then we construct an experiment in order to test our hypotheses.

Theory revision. Depending of the outcome of the experiment, we revise our theory in order to update our knowledge.

We distinguish between two cases, the outcome of the experiment will determine whether our current theory is too general, or too specific. Our theory is too general if it predicts an instance of the phenomenon that cannot appear in nature. On the other hand, our theory is too specific if it fails to predict all the instances of a given phenomenon. In the first case we need to specialize our theory, in the latter we need to generalise it. Thus, in each step we form a theory that explains more complicated events. This process is essentially inductive, the reasoning process that forms a general theory from a given (not exhaustive) set of particular examples. Induction is the reasoning process primarily employed in sciences, contrarily to the deductive reasoning, where we move from general assumptions to conclusions about specific instances, which is used in mathematics. A core achievement of ILP researchers at the end of the last century was to provide a comprehensive theory of reasoning that unifies inductive and deductive reasoning.

Figure 1 depicts a graphical representation of the above method. In this process we recognise three reasoning mechanisms, marked with dashed lines. On the one hand, a theoretical scientist will formulate a new hypothesis to test. Then an experimental scientist will construct an experiment, or a series of them, in order to verify the validity of the hypothesis. Then, again, the theoretical scientist will revise her theory based on the observations of the experiment. It is important to note that the nature is treated as a black box, whose internal mechanics we aim to discover. It is also important to distinguish between the hypothesis formulation and the experimentation. In general is not a trivial task to generate a series of experiments to test a hypothesis, since rarely the hypothesis can be tested in its entirety, more often we will be only able to test individual components of a hypothesis. Therefore, is also not trivial to revise our theory given the outcome of a series of experiments. In general we will distinguish between a formal language in which the hypotheses are formulated $\mathcal{H}$ and a language in which the experiments are formulated $\mathcal{E}$. Usually, we have $\mathcal{E} \subsetneq \mathcal{H}$, otherwise experimentation would be trivial. The nature can be formulated as an *oracle* function $\texttt{orac}(\cdot)$ on the experiment space. Observe that since we treat nature as simply an oracle function, "nature" may be referring to a variety of phenomena, a database, another programme[1], a electronic circuit etc.

[1] Observe that there are strong theoretical limitations, when applying ILP in order to explain another programme. Note that, in general, deciding whether two programmes are equivalent is *undecidable* [67]. This is important, since the output of an ILP algorithm explaining a programme P, will be a logic programme P' such that $P \approx P'$.

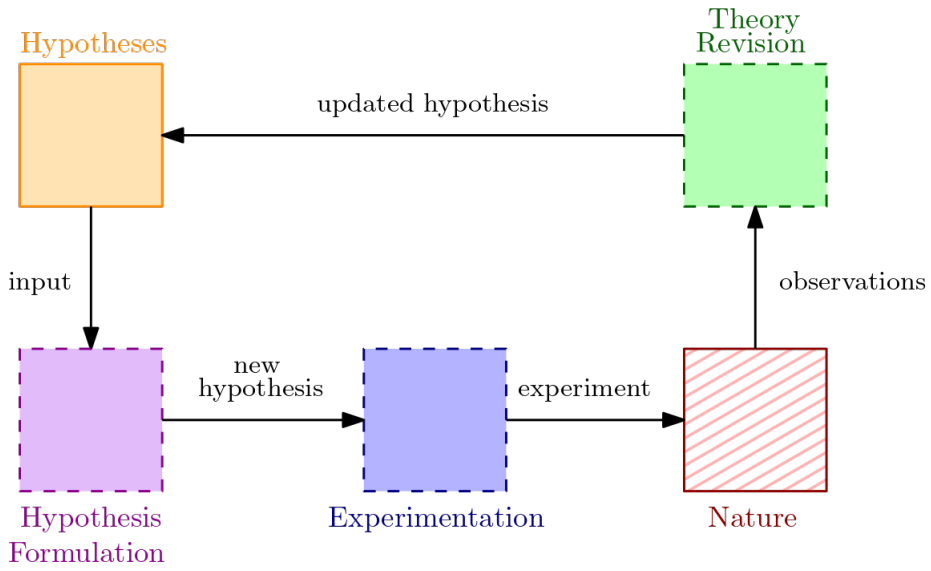


Figure 1: A graphical representation of the *scientific method*. The set of (current) *hypotheses* (H), contains all of our currents beliefs. Observe the three reasoning mechanisms marked with dashed lines. The *hypothesis formulation* (HF) mechanism chooses the next hypothesis to test, given the already known hypotheses. The *experimentation* (E) mechanism constructs an experiment (or a series of them) in order to verify the validity of the new hypothesis. The *theory revision* (TR) mechanism updates the hypotheses given the observations of nature on an experiment. The *nature* (N) is treated as an "black box".

Text Organisation

We begin by examining various theoretical frameworks for abstract inductive reasoning, in Chapter 1. Our goal in this chapter is to both examine some evaluation frameworks, such as *information theory* and *PAC-learnability*, and to establish a setting of abstract inductive reasoning based of partially ordered sets (posets). We will see there that elementary reasoning problems can be formulated as traversal problems in posets, where the nature can be viewed as a comparison oracle. We insist in this abstract approach based on posets, since it does not assume any particular representation of the underlying hypothesis space. Therefore, *theoretically* this approach can be applied in both symbolic and sub-symbolic learning approaches.

An important thing to note is that science proceeds with the belief (or work assumption) that an explanation always exists. In other words, that there exists a (perhaps semi) formal language that can describe the phenomenon. Note that this is not a trivial assumption. The hypothesis language influence both computational properties and expressivity of an inductive system. There is a trade-off between how expressive a formal hypothesis language is and how hard reasoning is in the related hypothesis space. In this paper we will consider symbolic hypotheses spaces, usually based on some logic formalism such as *propositional* or *first order* logic. In Chapter 2 we review some elementary results form classical mathematical logic. Our goal is to examine the mechanics and basic concepts of these formalisms. On the other hand, many computational problems related to first-order logic are computationally intractable. For this reason, the inductive logic programming community focused on a subset of first-order logic with better computational properties. This subset is *first-order Horn clauses*. The mathematical properties of the latter are presented in Chapter 3.

As we mentioned above, the scientific method can be seen as a *learning* procedure. The key difference between learning and scientific analysis lies in the notion of *explainability*. Humans are to "understand" a phenomenon in a purely intuitive or unconscious manner, without being able to explain that knowledge or transfer it to others. A scientist is obligated to provide an adequate explanation of the phenomenon and communicate this explanation to others. Therefore, in Chapters 4, 5, 6 we survey the area of *inductive logic programming*. We begin from *single clause induction* in Chapter 4. There, a hypothesis is constituted from a single Horn clause. In Chapter 5 we present methods for induction when a hypothesis is constituted by multiple Horn clauses. Learning a set of first-order Horn clauses is equivalent to inferring a logic programme, or a *theory*. Lastly, in Chapter 6 we study technical and algorithmic issues appearing in inductive

system implementation. We also survey some representative inductive systems, while we focus on query based approaches.

Lastly, in Chapter 7 we discuss our conclusions. We close this forward by arguing about the reasons that constitute inductive logic programming a fruitful area of research 30 years after its conception.

Motivation

The purpose of this text is to serve as a foundation for a doctoral research plan. We believe that the methods developed for inductive logic programming can be expanded to include other, perhaps even non-logical, methods of learning, thus serving as a unifying theory. By developing an abstract concrete theory of learning, focusing on its algorithmic aspects, in a query-based framework; will help us better understand the learning methods that exist today, but also may lead to innovative new approaches.

Since the advent of machine learning, many computational models have been proposed as a framework for studying the properties of a learning algorithm, forming the distinct research area of computational (or algorithmic) learning theory. Such a framework usually begins with a formal definition of learning. *What does it mean, for an algorithm to learn a concept C?* In traditional, sub-symbolic, "geometric" learning such frameworks are based on statistical assumptions. Roughly speaking, in these frameworks an algorithm learns a concept if it minimizes the mean error of its predictions. Perhaps the most popular of these approaches is the *PAC-learnability* (see Chapter 1). Other similar models include the Vapnik-Chervonenkis dimension (VC-dimension) and the mistake bound model [47]. These models provide both a concrete definition of *learnable* and theoretical tools for evaluating an algorithm's performance. The performance of a learning algorithm is assessed by its time and example complexity. In the era of "big-data" the latter aspect is of particular importance. Contemporary, deep learning systems are particularly data inefficient, which usually translates to considerable consumption of computational resources, and thus energy. Form a theoretical point of view perhaps the most elaborate of the statistical-based learning frameworks is Solomonoff's *theory of inductive inference* [22]. The latter is based on the notion of Kolmogorov complexity (see Chapter 1). Solomonoff's theory applicability is restricted by the fact that Kolmogorov complexity is undecidable [67].

In the realm of symbolic, logical learning there are parallel approaches to theoretical frameworks. The key difference between "symbolic computational learning" and its sub-symbolic counterpart is that the former frameworks do not depend on statistical assumptions. As we will see in Chapter 6 two main notions of learnability, are *identification in the limit* [30, 66] and *exact identification* [4]. The first model is of particular interest since it is the computational equivalent to the reputability principle of epistemology. If a learning algorithm identifies a theory to the limit, after a finite number of examples there is the guarantee that it will stop reconsidering its current theory. In other words, it would have reached the correct hypothesis. Unfortunately, the number of examples (or experiments) needed is indeterminable. Similarly, a learning algorithm exactly identifies a theory, if we have the guarantee that after the learning procedure is concluded, the resulting theory will be logically equivalent to the one held by nature. The relationship between PAC-learnability and ILP can be found in [16].

The computational models we mentioned above have mainly been developed at the end of last century drawing ideas form the computability and computational complexity theory, the latter being established at the middle of the 20th century. We believe that for the beginning of the 21st century there need to be established new frameworks responding to the key issues of contemporary learning methods, namely *data efficiency* and *explainability*.

We believe that the reasoning frameworks developed for ILP may provide an answer to both those issues. A key characteristic of the ILP methods is the reduction of the learning problem, to a search problem on a heavily structured hypothesis space; organized by a partial order. The elegant and robust algorithms developed for ILP, based on clause refinement with generalization and specialization operators, provide a clear view of the relationship between the hypothesis learned and the provided examples (see Chapters 1, 4, 5). In other words, the *effect* the provided data set has to an ILP algorithm is deterministic, and perhaps *invertible*. Namely, we *conjecture* that for each update to the current hypothesis of an ILP algorithm, the set of examples responsible for this change can be identified. Moreover, the effect of examples to an ILP algorithm may be *independent* of the entirety of the example set. Thus, for each new hypothesis, we can identify *deterministically*

a set of *witnesses*. This could lead to an alternate view on *explainability* of a learning system. A learning system following the properties highlighted here can be *"debuggable"*, meaning that we could omit examples that made the algorithm diverge to an unwanted hypothesis, or provide a well defined set of examples to guide the algorithm to a correct hypothesis.

On the other hand, recent advances in query-based poset sorting and searching algorithms [20, 5, 37, 9, 13], may lead to a better understanding of the relation between the quality of the learned hypothesis and the quantity (or quality) of examples. Since ILP methods reduce learning to poset search using queries, it would be interesting to compare learning algorithms to optimal searching, state-of-the-art query-based algorithms on posets. It is important to note that the work cited above focuses on the *query* complexity of the algorithms. Under the learning problem, the query complexity will correspond to the *example complexity*. As we shall see in Chapter 6, the main obstacle to the above approach is that the set of allowed queries differs in each area. In poset algorithms, an oracle answers comparison queries, while in ILP an oracle will answer membership and equivalence queries. Despite the intuitive connection between these two settings, the exact relation remains open.

From the above discussion it should be clear that we believe that the most fruitful area for research would be the connection between the inductive system and its environment, i.e. nature. The communication between an inductive algorithm and nature is materialized via experimentation, and the corresponding theory refinement, is dictated by the results of the experiments. Therefore, we focus on the blocks (E), (N), and (TR) of Figure 1. In the rest of this paper, we discuss elementary notions of ILP and related concepts, but also attempt to connect this presentation to our ideas for future research. We hope that this dual approach will inform the reader about the foundations of ILP, but also provide motivation for further exploration of this rich scientific area.

Acknowledgements

This paper has been submitted for the fulfillment of the General Postgraduate Examination, for doctoral students of the Computer Science Department, of University of Crete. In has been supported by the scholarship for graduate researchers of the Institute of Computer Science, of Foundation for Research and Technology – Hellas. The author would like to thank the post-doctoral researcher Constantinos Varsos and the principal researcher Giorgos Flouris for their guidance, comments and remarks during the writing of this paper.

February, 2024
Heraklion, Greece

Chapter 1

Preliminaries

In this chapter, we review some elementary notions that will prove useful in the sequel. Firstly, we begin with a brief introduction of *information theory*. Then, we discuss *partially ordered sets*. In the following chapters we will view hypotheses spaces organised as partially ordered sets, and more specifically *lattices*. In this section we, also, review some elementary problems such as *sorting* and *selection*. For these problems, in the query setting, we state some lower bounds on the number of queries, using information theory. Lastly, we discuss the main problem of this paper, namely *inductive* and *deductive* reasoning. We will present the basic notions in an abstract framework organised as a partially ordered set or a lattice. In this setting we introduce the elementary operators for traversing the hypothesis space.

1.1 Information Theory

We begin this chapter by presenting the elementary definitions and notions of Shannon's information theory [18]. A key measure in information theory is *entropy*. Entropy quantifies the amount of *uncertainty* involved in the value of an evolving structure, such as a random variable or a random process. In this section we follow the traditional approach regarding random variables, subsequently, in the next sections, we will generalize this concept to include other structures. We give the following definition.

Definition 1 (Entropy)**.** *Assume a random variable $X \in \mathcal{X}$, of a* finite *space $\mathcal{X}$, with probability mass function $p_X \colon \mathcal{X} \to [0,1]$. The entropy is a function that assigns a positive real value to any distribution on $\mathcal{X}$, i.e. $H \colon \Delta_{|\mathcal{X}|} \to \mathbb{R}_{\geq 0}$, where,*

$$H(X) = -\sum_{x \in \mathcal{X}} p_X(x) \log(p_X(x)). \tag{1.1}$$

In the above definition, with Δ_n we denoted the n-dimensional probability simplex, and with $\log(\cdot)$ we denote the base 2 logarithm. From an intuitive perspective, Shannon's entropy an object contains information *proportional* to its *unexpectedness*. In other words, the less likely that is an object to be observed, the more information is contained in its appearance. Additionally, Shannon's entropy regards a set of possible observation as a whole, thus measuring the *average* information transmitted by a sender, who sends symbols from $\mathcal{X}$ according to the distribution p_X. The maximum average entropy, per transition is achieved when the sender selects the symbols from $\mathcal{X}$ *uniformly*. For instance, from (1.1) we have that $H(X) = \log(n)$, when the random variable X follows the uniform distribution on a n-element sample space $\mathcal{X}$.

As we shall see in the sequel Shannon's entropy can provide tight bounds for simple query-based learning problems. In our setting entropy can be seen as a *measure*, or *lower bound*, on the number of "yes"-"no" queries needed, in order to determine the value of a random variable X given is probability mass function $p_X(\cdot)$. We give the following example.

Example 1 (Guessing Game)**.** *Assume that an adversary chooses a value $x \in \mathcal{X}$ according to a distribution $p_X(\cdot)$. An agent tries to guess the value x. The agent only knows the distribution $p_X(x)$. The agent is allowed to ask "yes"-"no"* membership queries *of the form $x \in A$, for $A \subseteq \mathcal{X}$. What is the optimal strategy the agent must follow in order to* minimise *the number of queries?*

Observe, that we can encode a series of answers in the above example as a bit-string, where "0" corresponds to "no" and "1" corresponds to "yes". When the sample space is finite, and for a rational agent, each string will be o finite length and at the end of the game the agent will have guessed the value x[1]. Thus the agents strategy induces an *encoding* of the sample space $\mathcal{X}$, since it relates each element of the sample space with a bit string. Thus, the objective of the agent is to minimise the *average length* $L(C)$ of this encoding. Such an optimal encoding can be achieved using the algorithm by Huffman [18]. It has been proved that it always holds $L(C) \geq H(X)$ and $H(X) \leq L(C_H) \leq H(X) + 1$ using the Huffman encoding C_H [18]. Thus, the notion of entropy constitutes a *tight* lower bound on the number of membership queries.

Observe that the Guessing Game of Example 1 is essentially a *learning problem*. The agent's goal is to learn the concept $x \in \mathcal{X}$. In order to achieve that the agent can make queries, thus forming examples of the form $\langle A, r \rangle$, where the query $x \in A$ is asked and the response r is given from the adversary. Additionally, in Example 1 it is assumed that the probability mass is known to the agent. This is no trivial assumption. In the sequel we will lift this assumption. We will assume that the agent has *no a priori* knowledge of the probability distribution. Instead the agent will utilise the structure of hypothesis space in order to achieve optimality.

1.1.1 Kolmogorov Complexity

Before we close this section we briefly mention an alternative view on information theory, this of Kormogorov complexity [41]. Here we present the notion of Kolmogorov complexity for strings, but it can easily be generalised to support any form of data, e.g. images or graphs. We give the following definition.

Definition 2 (Kolmogorov Complexity)**.** *Assume a binary language* $\mathcal{L} \subseteq \{0,1\}^\star$*. Then the* Kolmogorov complexity $K(\mathcal{L})$ *is the length of the binary description*[2] *of the smallest Turing machine that enumerates* $\mathcal{L}$*. In other words,*

$$K(\mathcal{L}) = \min\{|\langle M \rangle| \mid L(M) = \mathcal{L}\}. \tag{1.2}$$

Definition 2 captures the information hidden in a language $\mathcal{L}$ from a computational perspective. Intuitively, a complex object needs a more sophisticated algorithm to produce it. Naturally, for a finite language $\mathcal{L}$ we always have $K(\mathcal{L}) = O(|\mathcal{L}|)$, since the language is already a description of itself. Later in this paper we will use the notion of Kolmogorov complexity in order to find efficient explanations of examples. Informally, assume H to be an explanation of a set of examples E. Obviously, E is already an explanation of itself, thus, in order for H to be non trivial, we need $|H| < |E|$. While ideally we would like $|H|$ to approach $K(E)$. Unfortunately, due to the Halting Problem [67], we *cannot* compute $K(\mathcal{L})$ for any language $\mathcal{L}$, even if $\mathcal{L}$ is finite, or even if it is a singleton. Therefore, we can only heuristically determine the descriptive efficiency of our hypothesis H.

The connections and differences between the two alternatives take on the notion of information, i.e. Shannon's information theory and Kolmogorov's complexity are explored in [31]. Finally, we highlight that the notions form information theory briefly mentioned here have deep connections to formal learning theory. Kolmogorov complexity form a core notion of the *minimum description length* principle [32] which serves as a computational analogue to the central epistemology principle of Occam's razor. Moreover, Kolmogorov's complexity and Bayes' rule form play a vital role in Solomonoff's theory of inductive inference [22].

1.2 Partially Ordered Sets

In this section we discuss the structure that may be used by the agent instead of the a priori knowledge of a probability distribution, namely *partially ordered sets*, or *posets* for short [72]. We assume a set S and an order relations $\preceq$ on the elements of S. Depending on the properties of the relation $\preceq$, the structure $\langle S, \preceq \rangle$ will have different characteristics. In the following definition we provide some elementary properties our relation may have.

[1] We assume that a rational agent doesn't repeat the same question. A trivial strategy is to ask queries of the form $x \in \{\chi\}$, for each $\chi \in \mathcal{X}$. Following this strategy, in the worst case the agent will make $|\mathcal{X}|$ queries.

[2] The binary description $\langle M \rangle$ of a Turing machine, M is the description imported to the universal Turing machine U, such that $U(\langle M \rangle)$ simulates M.

Definition 3 (Order Relation Properties)**.** *Assume some set S, and a relation $\preceq \subseteq S \times S$. We define the following properties:*

- *We call the relation $\preceq$* reflexive*, when for each $s \in S$, $s \preceq s$.*
- *We call the relation $\preceq$* symmetric*, when for each $s, t \in S$, $s \preceq t \Leftrightarrow t \preceq s$.*
- *We call the relation $\preceq$* antisymmetric*, when for each $s, t \in S$, if $s \preceq t$ and $t \preceq s$ then $s \equiv t$.*
- *We call the relation $\preceq$* transitive*, when for each $s, t, u \in S$, if $s \preceq t$ and $t \preceq u$, then $s \preceq u$.*

In the above definition we use the notation $x \equiv y$ to describe *syntactic* equivalence, namely the fact that the variables x, y refer to identical syntactic representation[3]. When $x \preceq y$ we say that x is *dominated* by y, or that y *dominates* x. We call a relation that is reflexive, symmetric and transitive an *equivalence* relation. We call a reflexive and transitive relation a *quasi* order. Additionally, a relation is a *partial* order, when it is a quasi order and also antisymmetric. When a partial order relates *all* elements of S, then it is called a *total* order.

Assume a set S and a partial order relation $\preceq$ over S. The structure $\mathcal{P} = \langle S, \preceq \rangle$ is called a partially ordered set, or *poset*. A *minimum* element in $\mathcal{P}$ is some $m \in S$, such that, for all $s \in S$, it holds that $m \preceq s$. Symmetrically, a maximum element is some $M \in S$, such that, for all $s \in S$, we have $s \preceq M$. Note that a maximum or minimum element not always exists in a poset, since not all elements of S are related by $\preceq$. When it does, we write $m = \min \mathcal{P}$ and $M = \max \mathcal{P}$. Since the existence of $\min \mathcal{P}$ and $\max \mathcal{P}$ is not guaranteed we introduce some weaker notions of *minimality* and *maximality*. We say that an element m' is *minimal* when there exists no other element $s \in S \setminus m'$ such that $s \preceq m'$. Similarly, an element M' is *maximal*, when there is not an element $s \in S$, such that $M' \preceq s$. Minimal and maximal elements always exist in a poset. We denote with $\text{mil } \mathcal{P}$ and $\text{mal } \mathcal{P}$ the *sets* of minimal and maximal elements respectively.

Consider a sequence $s_1, s_2, \ldots, s_n$, where $s_1 \preceq s_2 \preceq \cdots \preceq s_n$. We call the sequence $s_1, s_2, \ldots, s_n$ a *chain* of the poset $\mathcal{P}$. Additionally, we call a sequence $s'_1, s'_2, \ldots, s'_n$, where there all elements of the sequence are *mutually uncompereable* by $\preceq$ an *antichain*. A *chain decomposition* is a partition of a poset into chains. The minimum number of chains in a chain decomposition of $\mathcal{P}$ is called the *width* of $\mathcal{P}$. A seminal result by Dilworth states that the width of a poset is equal to the length of the greater antichain [21]. The dual result by Mirsky states that the length of the greater chain is equal to the minimum number of antichains in an antichain decomposition of $\mathcal{P}$ [46].

An equivalence relation partitions S into equivalence classes. For example the modulo 3 relation partitions the set of integers into the sets $\langle 0 \rangle, \langle 1 \rangle, \langle 2 \rangle$. The equivalence class $\langle i \rangle$ contains all the integers than leave remainder i when divide with 3. Note that each integer belongs to one of the equivalence classes. It is often useful to assign *representatives* in each class. In the modulo 3 example the representative of each class is the smallest positive integer that belongs to the class. It is easy to see that, for $i \in [3]$ the representative of $\langle i \rangle$ is i itself.

In our context, equivalence classes are often useful to turn a quasi order into a partial order. Assume the set L_3 containing all lists with length at most 3, on the alphabet a, b, c, e.g. $[a, c] \in L_3$. Now, consider a relation $\preceq$, such that for $\ell_1, \ell_2 \in L_3$, it holds $\ell_1 \preceq \ell_2$, if $\ell_1 \subseteq \ell_2$. It is easy to see that $\preceq$ is a quasi order, namely reflexive and transitive. Unfortunately, $\preceq$ is *not* antisymmetric. Indeed let $\ell_1 = [a, b]$ and $\ell_2 = [b, a]$, then it holds that $\ell_1 \preceq \ell_2$ and $\ell_2 \preceq \ell_1$, but $\ell_1 \not\equiv \ell_2$. The problem is that ℓ_1, ℓ_2 are *syntactic variants* of the same set $\{a, b\}$. In order to overcome this difficulty we introduce an equivalence relation in L_3 where $\ell_1 \sim \ell_2$ when both lists correspond to the same set. As a representative of each class in L_3 we choose the list alphabetically sorted, namely the representative of the class $\langle [b, a] \rangle$ is $[a, b]$. Let $\langle L_3 \rangle$ be the set of representatives ordered with $\preceq$, now $\langle \langle L_3 \rangle, \preceq \rangle$ is a poset.

Posets will be of great importance in our study. Next we discuss a more restrictive type of poset, namely a lattice.

1.2.1 Lattices

A lattice is a poset $\mathcal{P} = \langle S, \preceq \rangle$, where for each two elements $s, t \in S$ there is a *greatest lower bound (glb)* and a *least upper bound (lub)*.

[3] For example $3 \equiv 3$, but $3 \not\equiv 2 + 1$, despite that $2 + 1 = 3$. The syntactic equivalence relation is a more primitive and weak relation than *semantic* equivalence, i.e. $2 + 1$ and 3 *mean* the same thing, but their syntax is different. We discuss the relation between syntactic and semantic equivalence in Chapter 2.

In order to formally define the notion of glb and lub, we need to introduce some notation. Assume the poset $\mathcal{P} = \langle S, \preceq \rangle$. Also assume some $s \in S$, with $\mathcal{P}_{\preceq x}$ we denote the sub-poset of all elements of $\mathcal{P}$ that are dominated by x. We also overload the set theoretic notation to posets, by assigning them the traditional semantics. A lattice can now be defined as a poset, where the operations *join* ($\wedge$), which gives the least upper bound, and *meet* ($\vee$), which gives the greatest lower bound, exist[4].

Definition 4 (Lattice)**.** *Let $\mathcal{L} = \langle S, \preceq \rangle$ be a poset. We call $\mathcal{L}$ a lattice, when for every $s, t \in S$ the following operations are defined.*

$$s \wedge t = \min \left[(\mathcal{L}_{s\preceq} \setminus s) \bigcap (\mathcal{L}_{t\preceq} \setminus t) \right] \tag{1.3}$$

$$s \vee t = \max \left[(\mathcal{L}_{s\succeq} \setminus s) \bigcap (\mathcal{L}_{t\succeq} \setminus t) \right] \tag{1.4}$$

A traditional example of a lattice includes the poset $\mathcal{L} = \langle 2^S, \subseteq \rangle$. There, the join relation corresponds to $\cup$, while the meet relation corresponds to $\cap$[5]. Here we will give a different example based on Number Theory.

Example 2 (Number Theory Lattice)**.** *Consider the set of numbers $N = \{3, 6, 9, 15, 18, 30, 45, 90\}$ ordered with the "divided-by" relation $\cdot|\cdot$*[6]*. The poset $\mathcal{L} = \langle N, | \rangle$ is actually a lattice, where the join operation $\wedge$ corresponds to the* least common multiplier*, while the meet operation $\vee$ corresponds to the* greatest common divisor*. In Figure 1.1 we present the* Hasse *diagram*[7] *of this lattice.*

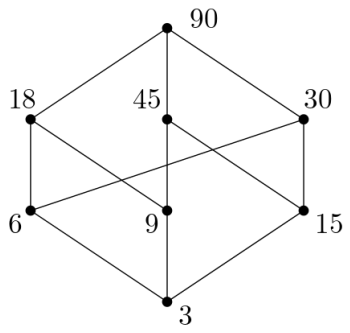


Figure 1.1: The Hasse diagram of Example 2.

It is possible to turn any poset into a lattice by incorporating the special elements *top* $\top$ and *bottom* $\bot$. Namely, assume a poset $\mathcal{P} = \langle S, \preceq \rangle$ where the join and meet operations are not defined for each pair $s, t \in S$. We expand $\mathcal{P}$ into a poset $\mathcal{L} = \langle S \cup \{\top, \bot\}, \preceq' \rangle$, where the relation $\preceq' \supset \preceq$ is the *smaller* extension such that the following holds for any $s, t \in S$:

$$s \wedge t = \begin{cases} \min \left[(\mathcal{L}_{s\preceq} \setminus s) \bigcap (\mathcal{L}_{t\preceq} \setminus t) \right], & \text{if it exists} \\ \top, & \text{otherwise} \end{cases} \tag{1.5}$$

$$s \vee t = \begin{cases} \max \left[(\mathcal{L}_{s\succeq} \setminus s) \bigcap (\mathcal{L}_{t\succeq} \setminus t) \right], & \text{if it exists} \\ \bot, & \text{otherwise} \end{cases} \tag{1.6}$$

While expanding a poset into a lattice using equations (1.5), (1.6) provides no additional information, it is often useful for technical purposes and we will be using it in the sequel. In the next section we show how the structure we developed here can be used to model elementary reasoning processes.

[4] Here we diverge form the traditional notation. Usually, in Lattice theory the join operation is denoted by $\vee$, while the meet relation by $\wedge$. The reason for our divergence will become apparent in Chapter 2, where the join, meet operators will coincide with the logical operators $\wedge, \vee$. Have we followed the traditional notation the latter would not happen, thus creating confusion to the reader (as it did to the author, while studying the literature).

[5] Here the traditional notation would be more intuitive.

[6] We say that "*a divides b*", and write $a|b$, when there is some $c \in \mathbb{N}$, such that $b = c \cdot a$.

[7] A Hasse diagram is a way to represent a poset graphically. In this graph we omit transitive edges, while we assume an implicit orientation of the edges from top to bottom.

1.3 The Deductive & Inductive Reasoning Process

There are two elementary reasoning processes that an intelligent agent should be able to perform, *deduction* and *induction*. Deduction is the ability to draw conclusions from existing knowledge, while induction is the process of *generalising* and forming new knowledge from particular examples. Note that induction is essentially a *learning* procedure. In this section we show how these reasoning mechanism can simply be viewed as traversal problems on a lattice[8]. We discuss deductive reasoning first. The ideas discussed here come from [61].

1.3.1 Deductive Reasoning

Assume a hypothesis language $\mathcal{H}$ and a query language $\mathcal{Q}$. For now, we treat the languages $\mathcal{H}, \mathcal{Q}$ as simple sets. Additionally, assume that $\mathcal{H} \cup \mathcal{Q}$ are organised by a *generality* relation $\succeq$, where $g \succeq s$, when g is a more general concept than s. Here we will also treat the relation $\succeq$ abstractly. Intuitively $\succeq$ is an *explanation* relation, i.e. when we say that g is more general than s, $g \succeq s$, we mean that g *explains* s. Now the deductive reasoning problem can be stated as poset traversal problem in $\mathcal{P} = \langle \mathcal{H} \cup \mathcal{Q}, \succeq \rangle$[9]. Formally, deduction can be stated as following.

GeneralDeduction (D)	
Input:	1. A knowledge base $H \subseteq \mathcal{H}$. 2. A query $q \in \mathcal{Q}$. 3. A generality relation $\succeq$ on $\mathcal{H} \cup \mathcal{Q}$.
Output:	▶ "YES", if there is a chain $h \equiv s_1, s_2, \ldots, s_n \equiv q$ in $\langle \mathcal{H} \cup \mathcal{Q}, \succeq \rangle$, s.t. $h \in H$. ▶ "NO", otherwise.

Note that in the above problem definition we assume that we have explicit access only to a subset H of the hypothesis language $\mathcal{H}$. Nevertheless, our goal is to find whether q is accessible by a chain $s_1, s_2, \ldots, s_n$ that may include elements not included in out knowledge base H. Thus a mechanism must be provided that *generates* these intermediate steps. In other words, we aim to traverse a poset $\mathcal{P}$, when only a small part of it is given. The reason for this limitation is practical, since the hypothesis language $\mathcal{H}$ might be *infinite*, while it is reasonable to assume that, the given knowledge will be finite. Therefore, becomes apparent that the "NO" answer in the above statement *depends* on the available knowledge; there might be a chain from some $h \in \mathcal{H} \setminus H$ to q, but the system will be unable to retrieve it. In Figure 1.2 we present both feasible and infeasible instances of the GeneralDeduction problem.

(a) A feasible example of the General Deduction problem

(b) A infeasible example of the General Deduction problem

Figure 1.2: Some possible instances of the General Deduction problem.

[8] Interestingly enough, the idea we highlight here follows closely a general principal of mathematical logic. Namely, *"any logic corresponds to a lattice, and vice versa"*, under some reasonable assumptions [25].

[9] Note that here we diverge form the traditional presentation of deduction in formal systems (e.g. followed in [61]), in the sense that we regard the query space $\mathcal{Q}$ and the hypothesis space $\mathcal{H}$ organised by the same relation $\succeq$. As we shall see in Chapter 3 we may consider the programme $P = \{a, b \leftarrow a\}$. A possible query could be $q \equiv b \leftarrow$, which queries whether $P \models b$. In logic programming we answer such queries by *resolution* (see Chapter 3). We test if by appending the query to the programme will result in contradiction, i.e. $(P \cup q) \models \bot$. Equivalently, we could ask if there is some programme P', such that $P \models P'$ and $(b \leftarrow) \in P'$. Therefore our alternative representation is without loss of generality.

1.3.2 Inductive Reasoning

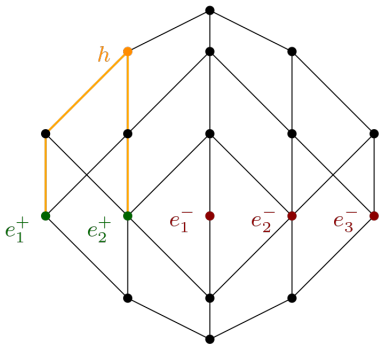
Inductive reasoning is slightly more complicated than deduction. Additionally we recognise multiple variants of the same problem, which we state below. Again, we assume a hypothesis language $\mathcal{H}$. Instead of the query language $\mathcal{Q}$, we consider a language $\mathcal{E}$ of examples. Again, we assume an underlying poset $\mathcal{P} = \langle \mathcal{H} \cup \mathcal{E}, \succeq \rangle$, with $\succeq$ being, again, a generality-explanation relation. Below, we formally state the GeneralInduction-SingleHypothesis problem, where we want to cover each example with a *single* hypothesis.

GeneralInduction-SingleHypothesis (IS)	
Input:	1. A set of *positive* examples $E^+ \subseteq \mathcal{E}$. 2. A set of *negative* examples $E^- \subseteq \mathcal{E}$. 3. A generality relation $\succeq$ on $\mathcal{H} \cup \mathcal{E}$.
Output:	▶ "YES", if there is an $h^\star \in \mathcal{H}$, s.t.: a) For each $e^+ \in E^+$, there is a chain $h^\star \equiv s_1, s_2, \ldots, s_n \equiv e^+$ in $\langle \mathcal{H} \cup \mathcal{E}, \succeq \rangle$. b) There is no $e^- \in E^-$, s.t. $h^\star \succeq e^-$. ▶ "NO", otherwise.

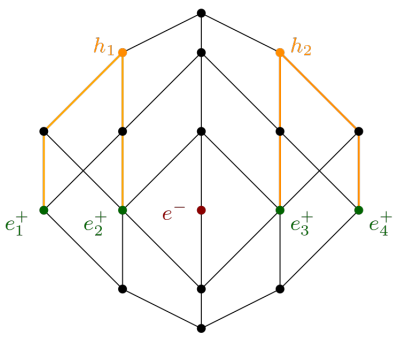
Naturally, we may allow the set of examples to be *covered* by multiple hypotheses. We say that a hypothesis h *covers* an example e, when $h \succeq e$, we denote this with $c(h) = \{e \in \mathcal{E} \mid h \succeq e\}$. Observe that, due to *transitivity* the existence of a chain from h to e is equivalent to $h \succeq e$. Assume a hypothesis $h \in \mathcal{H}$. We say that h is *consistent*, when $c(h) \supseteq E^+$. Additionally, we call h *complete* when $c(h) \cap E^- = \varnothing$. Of course, we aim for a both consistent and complete hypothesis h. Below we state the GeneralInduction-MultiHypotheses variant.

GeneralInduction-MultiHypotheses (IM)	
Input:	1. A set of *positive* examples $E^+ \subseteq \mathcal{E}$. 2. A set of *negative* examples $E^- \subseteq \mathcal{E}$. 3. A generality relation $\succeq$ on $\mathcal{H} \cup \mathcal{E}$.
Output:	▶ "YES", if there is an $H^\star \subseteq \mathcal{H}$, s.t.: a) For each $e^+ \in E^+$, there is an $h \in H^\star$, s.t.: there is a chain $h \equiv s_1, s_2, \ldots, s_n \equiv e^+$ in $\langle \mathcal{H} \cup \mathcal{E}, \succeq \rangle$. b) There is no $e^- \in E^-$, s.t.: $h \succeq e^-$, for some $h \in H$. ▶ "NO", otherwise.

The GeneralInduction-MultiHypotheses problem is a proper generalization of the GeneralInduction-SingleHypothesis problem, since there may be instances of positive-negative examples E^+, E^-, that cannot be covered by a single hypothesis, but can be covered by multiple hypotheses. In Figure 1.3 we present an instance of the GeneralInduction-SingleHypothesis and an instance of the GeneralInduction-MultiHypotheses problem that cannot be covered from a single hypothesis.



(a) An example of GeneralInduction-SingleHypothesis problem.



(b) An example of GeneralInduction-MultiHypothesis problem.

Figure 1.3: Example instances for induction problems.

Induction with Background Knowledge

We close the discussion in this section with a note regarding the *a priori* knowledge in the induction problem. From a practical point of view it is often useful to assume some *background* knowledge, when trying to generalise a set of examples. This reduces the computational cost for finding a complete set of hypotheses. In this direction, we assume a language $\mathcal{B} \subseteq \mathcal{H}$, and a given set $B \subseteq \mathcal{B}$ of background knowledge. Again, in this chapter we treat the above languages as abstract sets of elements, organised with a partial order $\succeq$. In this variant, we aim to expand B into some $H \supseteq B$, that covers all of the positive elements in E^+ and none of the negative examples in E^-. Below we expand the definition of the multiple hypotheses induction problem, adding background knowledge.

GeneralInduction-MultiHypotheses-BK (IMB)	
Input:	1. A set of *positive* examples $E^+ \subseteq \mathcal{E}$. 2. A set of *negative* examples $E^- \subseteq \mathcal{E}$. 3. A set of background knowledge $B \subseteq \mathcal{B} \subseteq \mathcal{H}$. 4. A generality relation $\succeq$ on $\mathcal{H} \cup \mathcal{E}$.
Output:	▶ "`YES`", if there is an $H^\star \supseteq B$, s.t.: a) For each $e^+ \in E^+$, there is an $h \in H^\star$, s.t.: there is a chain $h \equiv s_1, s_2, \ldots, s_n \equiv e^+$ in $\langle \mathcal{H} \cup \mathcal{E}, \succeq \rangle$. b) There is no $e^- \in E^-$, s.t.: $h \succeq e^-$, for some $h \in H$. ▶ "`NO`", otherwise.

Note for the corresponding GeneralInduction-SingleHypotheses-BK, we assume $|H^\star \setminus B| \leq 1$.

1.3.3 Abstract Reasoning Framework

We close this section with a definition of an *abstract reasoning framework*. From the above discussion it should be obvious that deduction and induction share many similarities. As it turns out they can be treated under the same abstract model. Note that in induction, the query language of deductions is essentially replaced by the examples language. Additionally, in an incremental setting we can treat IMB and IM the same. For multiple hypotheses we gradually create a sequence $H_1 \subseteq H_2 \subseteq \cdots \subseteq H_n$, where we expand H_i to H_{i+1} in order to cover a new subset of positive examples. Thus, we may assume $B = H_1$ is such a sequence. Lastly, we will consider $\mathcal{E} \subseteq \mathcal{H}$ and $\mathcal{B} \subseteq \mathcal{H}$. Therefore, an abstract reasoning framework will be essentially a poset.

Definition 5 (Abstract Reasoning Framework)**.** *Consider a hypothesis language $\mathcal{H}$, and a partial order $\succeq$ on $\mathcal{H}$. An* abstract reasoning framework *is the poset $\mathcal{R} = \langle \mathcal{H}, \succeq \rangle$.*

In the following chapters, we will examine different reasoning frameworks depending on the choice of hypothesis language $\mathcal{H}$ and a generalization order $\succeq$. Deciding which subset of the hypothesis language will form the example language $\mathcal{E}$ or the language for background knowledge $\mathcal{B}$ constitutes primarily an *implementation* issue and will be discussed when we examine various implementations of inductive systems. In the next section, we deal with some algorithmic aspects of our framework regarding the traversal in the hypothesis space.

1.4 Traversing the Hypotheses Space

After organising the search space we need a way to traverse it. In this direction we present two such families of *refinement* operators, namely *generalization* and *specialization* operators. Again, we follow the presentation of [61]. Assume an abstract reasoning framework $\mathcal{R} = \langle \mathcal{H}, \succeq \rangle$. A refinement operator is defined as $\rho \colon \mathcal{H} \to 2^{\mathcal{H}}$. As its name suggests, the generalization operator is a selection over the generalizations of a set of hypotheses H. Namely,

$$\text{for each } h \in \mathcal{H} \text{ we have } \rho_g(h) \subseteq \{h' \mid h' \succeq h\}. \tag{1.7}$$

A specialization operator is defined dually, as a selection over the specialization of H. That is,

$$\text{for each } h \in \mathcal{H} \text{ we have } \rho_s(h) \subseteq \{h' \mid h' \preceq h\}. \tag{1.8}$$

We now consider two important classes of refinement operators, namely *ideal* and *optimal* operators. First, we introduce some notation, for a partial order relation $\preceq$, we introduce the relation $\sqsubset \subseteq \preceq$, where $a \sqsubset b$, when $a \preceq b$ and b is the *smallest* element, with respect to $\preceq$, with this property. The *ideal* operator returns the *immediate* generalizations or specializations of a hypothesis h. Formally, we have the following.

$$\rho_{i,g}(h) = \{h' \mid h' \sqsupset h\}$$
$$\rho_{i,s}(h) = \{h' \mid h' \sqsubset h\}$$

Observe that we can explore the whole hypotheses space using an ideal operator. Indeed from $\sqsubset$ we can *reconstruct* $\preceq$, since the latter relation constitutes the *transitive closure* of the former. On the other hand there may be duplicate terms, since $\rho_{i,g}(h_1) \cap \rho_{i,g}(h_2) \neq \varnothing$. In order to avoid such redundancies, we introduce the optimal operator. In a lattice structure an optimal operator creates a unique path from the top element $\top$ to each hypothesis $h \in \mathcal{H}$. Thus, an ideal operator can reconstruct the Hasse diagram of a partial order, while an optimal operator only creates a tree.

Lastly, when $\mathcal{R}$ has the additional structure of a *lattice*, we are able to use the join $\wedge$ (see 1.3) and meet $\vee$ (see 1.4) operators defined in Section 1.2. In the inductive reasoning framework the join operator is called *least general generalization*, while the meet operator is called *most specific specialization*.

1.4.1 Refinement Operators in Multi-Hypothesis Setting

Now we discuss refinement operators in the multiple hypothesis setting. Given a refinement operator ρ for a single hypothesis, we can construct refinement operators that manipulate sets[10] as follows.

$$\gamma_{\rho,g}(H) = \{(H \setminus h) \cup h' \mid h' \in \rho_g(h), h \in H\} \cup \{H \cup h \mid h \in \mathcal{H}\} \tag{1.9}$$
$$\gamma_{\rho,s}(H) = \{(H \setminus h) \cup h' \mid h' \in \rho_s(h), h \in H\} \cup \{H \setminus h \mid h \in H\} \tag{1.10}$$

Equations (1.9), (1.10) should be intuitively clear. In (1.9), we either generalize one of the hypotheses of H, or add a new hypothesis h. On the other hand, in (1.10) we are allowed to either specialize one of the hypotheses in H, or remove one.

1.5 From Examples to Queries

Thus far we assumed that the positive and negative examples are already available to an inductive reasoner. In this section we consider an alternative model, where an inductive system has only access to an abstract oracle `orac`: $\mathcal{E} \to \{+, -\}$, which given an element e of the language $\mathcal{E}$, informs the reasoner whether e should be covered by a hypothesis $h \in \mathcal{H}$ or not. In the rest of this section we will examine firstly, two simple examples. The first regards a *totally ordered* hypotheses space, while the second a *partially ordered* space. Later, we review a related work by C. Daskalakis et al. [20] regarding an alternative oracle based problem for sorting in a poset. Our goal is to draw some similarities between inductive reasoning, and sorting problems in a poset. Based on these similarities we state a conjecture on a lower bound of the number of queries needed to find an optimal hypothesis in an abstract reasoning framework.

1.5.1 Induction in a Totally Ordered Hypothesis Space

Here we discuss a simple example drawn from traditional machine learning, namely *linear classification* [71]. Consider the euclidean semi-plane $\mathbb{R}^2_{y\geq 0}$[11]. Also, assume that our hypothesis space is a *finite* family of semi-lines $\mathcal{H} = \{h_1, h_2, \ldots, h_n\} \subset \mathbb{R}^2_{y\geq 0}$ intersecting $(0, 0)$. We encode each semi-line in $h \in \mathcal{H}$ with its unimodular, collinear vector $\vec{v}_h$. In the sequel we make no separation between $h, \vec{v}_h$. Let the example language consist of all the points in the semi-plane, i.e. $\mathcal{E} = \mathbb{R}^2_{y\geq 0}$. We say that $h \in \mathcal{H}$ *explains* an example $e \in \mathcal{E}$, when e falls on the *counter-clockwise* side of h.

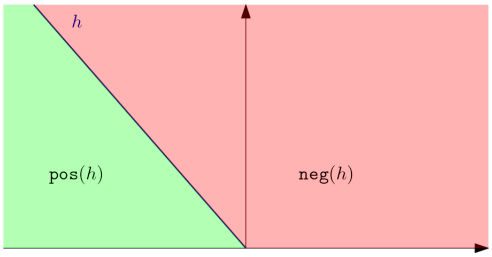
[10] In literature, often a set of hypothesis is referred to as a *theory*. Thus, the operators introduced here are sometimes called *theory refinement operators*.

[11] Using exactly the same method, we could expand this example to the whole $\mathbb{R}^2$, but the proof of correctness involves some elementary concepts of analytic geometry that fall outside of the scope of this paper.

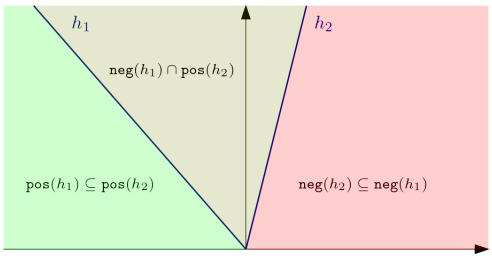
In the 2-dimensional space the *counter-clockwise* predicate [7] can be computed by the sign of the determinant of the two vectors, i.e.:

$$\texttt{CCW}(h, e) = \text{sign} \begin{vmatrix} h_1 & e_1 \\ h_2 & e_2 \end{vmatrix} \tag{1.11}$$

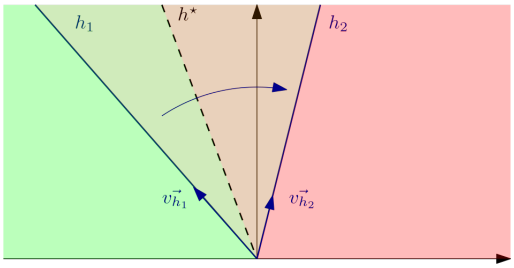
Thus $\texttt{CCW}(h, e) = +$, if e is on the counter-clockwise side of h, $\texttt{CCW}(h, e) = -$ if e is on the clockwise side of h, and $\texttt{CCW}(h, e) = 0$, when h, e are *collinear*. For some $h \in \mathcal{H}$, we denote with $\texttt{pos}(h), \texttt{neg}(h)$ the counter-clockwise, and clockwise sides of the semi-line respectively (see Figure 1.4a). We assume that $\mathcal{H}$ is *totally ordered* with respect to $\texttt{CCW}(\cdot, \cdot)$, i.e. $h_1 \preceq h_2$, when $\texttt{pos}(h_1) \subseteq \texttt{pos}(h_2)$ and $\texttt{neg}(h_1) \supseteq \texttt{neg}(h_2)$[12] (see Figure 1.4b).



(a) The separation of $\mathbb{R}^2_{y\geq 0}$ from a hypothesis $h \in \mathcal{H}$ into $\texttt{pos}(h), \texttt{neg}(h)$.



(b) The relation between two hypotheses, where $h_1 \preceq h_2$.



(c) One step of our binary sort variant for solving a simple induction problem based on queries.

Figure 1.4: Induction in a totally ordered hypotheses space.

Consider that an *adversary* has partitioned $\mathbb{R}^2_{y\geq 0}$ into E^+ and E^-, of positive and negative examples, with respect to a semi-line $h^\star$. We are given access to an oracle function, $\texttt{orac}\colon \mathcal{E} \to \{+, -\}$, where we assume that the adversary answers truthfully, i.e. $\texttt{orac}(e) = +$, if and only if $e \in E^+$ in her partition. Our goal is to find the smaller $h \in \mathcal{H}$, with respect to $\succeq$, such that $\texttt{pos}(h) \supseteq E^+$[13], using *the least* oracle calls, or *queries*.

Observe that the covering relation is *monotonic* with respect to $\preceq$. In other words, when h does not cover the set of positive examples, i.e. $\texttt{pos}(h) \not\supseteq E^+$, then *none* of the hypotheses h', such that $h' \preceq h$, will explain E^+. With this in mind, we are able to construct a *naive* algorithm which traverses from the minimum hypothesis h_1 to h_n. For each h_i we make the query $\texttt{orac}(\vec{h_i})$ to the adversary. If $\texttt{orac}(h_i) = -$, we stop and return h_{i-1}, otherwise we *generalise* to h_{i+1}. The algorithm stops when it reaches the first hypothesis that covers E^+. It is easy to argue that the hypothesis $\hat{h}$ is the *best approximation* of $h^\star$ in $\mathcal{H}$, provided that there is such $\hat{h} \in \mathcal{H}$, such that $\texttt{pos}(\hat{h}) \supseteq E^+$. This, naive, algorithm makes, in the *worst case* $O(n)$ queries, where $n = |\mathcal{H}|$. Can we do any better?

Consider the *median* hypothesis h_m of $\mathcal{H}$ with respect to $\preceq$. If $\texttt{pos}(h_m) \not\supseteq E^+$ then *none* hypothesis $h \in \mathcal{H}$ such that $h \preceq h_m$ will explain the positive examples. Thus, we may discard all hypotheses $\mathcal{H}_{\preceq h_m} = \{h \in \mathcal{H} \mid h \preceq h_m\}$. We repeat this process with $\mathcal{H} \setminus \mathcal{H}_{\preceq h_m}$ as our hypothesis space. This observation results in a variant of *binary search* for inductive reasoning in a totally ordered hypotheses space (see Figure 1.4c). In each recursive step, we consider the median h'_m of the *current* hypotheses space $\mathcal{H}'$. If $\texttt{orac}(h'_m) = +$, we specialise and continue our search in $\mathcal{H}'_{\preceq h_m}$. Otherwise, we need to generalise, and continue our search in $\mathcal{H}' \setminus \mathcal{H}'_{\preceq h_m}$. We conclude our

[12] In our example, the order we assumed corresponds to a *decreasing* order of the angles of the hypotheses $\vec{h}$ with the $x'x$ axis.

[13] Using the terminology introduced in Section 1.3 we demand the hypothesis to be *consistent*, but not necessarily *complete*. This is because we didn't force the adversary to choose $h^\star \in \mathcal{H}$.

search when the current hypotheses space $\mathcal{H}'$ becomes a *singleton*, i.e. $\mathcal{H}' = \{\hat{h}\}$. We return $\hat{h}$. The last query we made is $\texttt{orac}(\hat{h})$. If $\texttt{orac}(\hat{h}) = -$, we would specialise (if we had any hypotheses left). In this case $\texttt{pos}(\hat{h}) \supseteq E^+$, and our hypothesis satisfies our criteria. On the other hand, if $\texttt{orac}(\hat{h}) = +$ we would generalise. In this case, we will have $\texttt{neg}(\hat{h}) \cap E^- \neq \varnothing$ and $\hat{h}$ will be the *greatest* hypothesis that does not explain any of the negative examples. In the latter case, the hypothesis will not meet our criteria, but we could argue *intuitively* that, in some sense, this is the best we could do provided the set of hypotheses $\mathcal{H}$.

It is easy to see that the above method makes only $O(\log n)$ queries, in the worst case, since in any step the current hypothesis space gets halved. Observe that the above example is quite simple since essentially $\mathcal{H} = \mathcal{E}$, both hypotheses *and* examples are vectors in $\mathbb{R}^2_{y\geq 0}$. Additionally, the hypothesis of the adversary is also a vector in the same space. This will not hold in general. Additionally we considered a refinement of the IMB problem, where we allow the hypothesis h to explain some of the negative examples E^-, and to do not explain some of the positive examples E^+. Additionally, it is *crucial* to our analysis that the set of hypotheses $\mathcal{H}$ is *finite*. We believe that the above method is *optimal* and *general* for a totally ordered reasoning framework. We make the following conjecture.

Conjecture 1 (Lower Bound: Queries in Totally Ordered Reasoning Framework)**.** *Assume a language $\mathcal{H}$ of hypotheses and $\mathcal{E}$ of the examples, and a formal language $\mathcal{L} \supseteq \mathcal{H} \cup \mathcal{E}$, and a pair of positive and negative examples $E^+, E^- \subseteq \mathcal{E}$, respectively. Consider the oracle function $\texttt{orac}\colon \mathcal{E} \to \{+, -\}$, that answers truthfully for an unkown adversary's hypothesis $h^\star$. Also, let $\mathcal{R} = \langle \mathcal{L}, \preceq \rangle$ be a* totally ordered *reasoning framework. There is no algorithm that finds $h^\star$ with less than $O(\log n)$ queries in the worst case*[14]*.*

In order to prove the above conjecture, we need to find an instance for every possible algorithm that will force it to make $O(\log n)$ queries. An interesting direction that is often employed in such proofs is to devise a truthful algorithm for the *adversary* that will force any algorithm to make $O(\log n)$ queries. Subsequently, we present a similar conjecture for partially ordered reasoning frameworks. Before that, we discuss a related work by Daskalakis et al. [20] regarding sorting problems in posets.

1.5.2 Query Optimal Sorting in Posets

One of the most well studied problems in Computer Science is *Sorting* and *Searching*. Nevertheless, the main focus of the related research considers totally ordered sets. Interestingly enough, many important questions regarding the same problems in a partially ordered setting remained open until recently. In this Section we review a query optimal method for sorting a poset, published by Daskalakis et al. in 2011 [20]. A more gentle introduction to poset sorting using queries [20, 5, 9, 13] can be found in [56].

Before we present the algorithm, it is important to clarify the setting. Essentially, in a sorting problem our goal is to reconstruct an order relation on a given finite set. In the traditional, comparison-based setting we are only allowed to compare pairs of elements. Thus, we aim to reconstruct an order relation $\preceq$, by asking queries of the form $c(a, b)$. The oracle will answer with one of the following $\preceq, \succ, \not\sim$, whether the a is dominated by b, b is dominated by a, or the two elements are not related, respectively. We present the PosetSorting problem formally bellow.

PosetSorting	
Input:	1. A partially ordered set $\mathcal{L}$. 2. An oracle function $c\colon \mathcal{L} \times \mathcal{L} \to \{\preceq, \succ, \not\sim\}$, of a partial order relation $\preceq$ on $\mathcal{U}$. 3. A constant $w \in \mathbb{N}$, with $\text{width}(\mathcal{L}, \preceq) = w$.
Output:	The partial order relation $\preceq$.

In order to construct a *query-optimal* algorithm for the above problem, the authors in [20] devised a method called EntropySort. Consider a totally ordered setting. Assume the following

[14] It is crucial to note the quantifiers in the above conjecture. Assume a family of algorithms $\mathcal{A}$, for a family of instances $\mathcal{I}$ of a given problem. We say that $q(n)$ is a *lower bound*, when, there is no $A \in \mathcal{A}$, such that for every $I \in \mathcal{I}$, with $n = |I|$, consume less than $q(n)$ resources. In other words, for every algorithm $A \in \mathcal{A}$, there is an instance $I \in \mathcal{I}$, with $n = |I|$, such that A with input I consumes more that $q(n)$ resources.

incremental variant of merge-sort. In the i-th step, we keep an ordered subset $\mathcal{L}_i \subseteq \mathcal{L}$. At the beginning we have $\mathcal{L}_0 = \varnothing$. In order to construct $\mathcal{L}_{i+1}$ we need to sort a yet unsorted element $\ell \in \mathcal{L} \setminus \mathcal{L}_i$. We do this by performing a *binary-search* on $\mathcal{L}_i$ in order to find the correct position of ℓ.

The authors in [20] followed the same principles, in EntropySort, as our incremental variant of merge-sort[15] above. In each iteration we keep a *minimum chain decomposition* of the current sorted poset. In order to increment the current poset, we sort, a yet unsorted, element, by performing a binary search on each of the chains of the chain decomposition. In the new incremented poset, we reconstruct a minimum chain decomposition[16]. We repeat these steps until the poset is sorted.

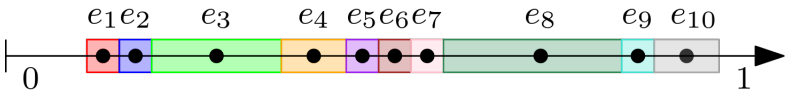


Figure 1.5: The partition of the $[0, 1]$-interval with respect to the elements of the i-th chain.

In order to achieve query optimality, we need to employ a variant of binary-search, called *weighted binary-search*. We will map the elements of a chain to sub-intervals of $[0, 1]$ (see Figure 1.5). The length of each sub-interval will be *proportional to the number of candidate posets that will be eliminated after the insertion of the element*. In other words, we use fewer queries for insertions that are less informative. Let $\mathcal{C} = \{C_1, \ldots, C_w\}$ be a minimum chain decomposition. We need to sort a new element e with respect to each chain $C_i \in \mathcal{C}$. In order to achieve this, in a partially ordered setting, we need to find an element e_i^+ that is the *smallest element that dominates* e, and an element e_i^- that is the *greatest element dominated* by e, for each chain $C_i \in \mathcal{C}$. We firstly consider the steps for finding e_i^+. Assume the element $e_{ik} \in C_i$. With $\mathcal{D}_{ik}$ we denote the number of possible w-width extensions of the current where e_{ik} is *the smallest element that dominates* e. Additionally, for the i-th chain with $\mathcal{D}_i$ we denote the number of all possible w-width extensions, i.e. $\mathcal{D}_i = \sum_{i=1}^{w} \mathcal{D}_{ik}$. The weight, or *mass* that we will assign to each element of the i-th chain is given by the following ratio,

$$\texttt{mass}_{i,e}^{+}(e_{ik}) = \frac{\mathcal{D}_{ik}}{\mathcal{D}_i}. \tag{1.12}$$

The number of queries needed in order to find e_i^+ is given in the following lemma.

Lemma 1 (Daskalakis et al. 2011 [20])**.** *Assume the i-th chain C_i of a minimum chain decomposition. Also, consider an element $e \notin C_i$ to be sorted. The smallest element e_i^+ of C_i that dominates e is found after* at most

$$2\left(1 + \log \frac{1}{\texttt{mass}_{i,e}(e_i^+)}\right) \tag{1.13}$$

Essentially in Lemma 1 we saw that we need $O(-\log(\texttt{mass}_{i,e}^{+}(e_i^+)))$ queries. Note the similarity between the equations (1.13) and (1.1). Observe that uninformative insertions would correspond to large mass. namely, if some element e_{ik} has large mass, then we would have that e_{ik} will be smallest element of C_i that dominates e in "many universes". Since it is likely that e_{ik} is the smallest dominator of e verifying this fact does not tells us much about the structure of the unknown poset. This observation, also, explains the name EntropySearch. We work symmetrically in order to find the greatest element e_i^- that is dominated by e. The query complexity is given in the following theorem.

Theorem 1 (Daskalakis et al. 2011 [20])**.** EntropySearch *sorts any finite partially ordered set of width w on n elements using $\Theta(n \log n + wn)$ queries.*

It turns out that EntropySort is *optimal* with respect to queries. On the other hand, EntropySort needs *exponential time*. The bottleneck lays on the computation of the number of possible w-width extensions $\mathcal{D}_{i,k}$. Whether there exists an efficient query-optimal algorithm remains an open question. In [20] the authors present also Bin-Insertion Sort and MergeSort whose time and query complexity are given in Table 1.1.

[15] Perhaps the term "merge-sort" is not the proper one to describe sorting via iterative binary-search, since nothing is "merged". Nevertheless, traditional merge-sort is equivalent to our variant computationally. We use this term, somewhat arbitrarily, in order to ease terminology.

[16] In general, a minimum chain decomposition must be computed from scratch in each iteration. We cannot just insert the new element in some of the chains and maintain a minimum decomposition.

	Query Complexity	Time Complexity	Lower Bound
Bin-Insertion Sort	$O(wn \log n)$	$O(n^4 - n^2w^2 + w \log n)$	$\Omega(n(\log n + w))$
Entropy Sort	$O(n \log n + wn)$	Exponential	
Merge Sort	$O(wn \log(n/w))$	$O(w^2 n \log(n/w))$	

Table 1.1: The algorithms of [20] and their respective *query* and *time* complexity. w is an upperbound to the width of the unknown poset. In the fourth column, we show the information-theoretic lower bound for the poset-sorting problem. Note that the query lower bound is also lower bound for the time complexity.

1.5.3 Induction in Partially Ordered Hypotheses Space

In this Section we discussed the query-based induction model through examples. In this setting we need to find an unknown hypothesis (held by an adversary) in order to explain a phenomenon. In order to do this, we are able to perform experiments (or ask queries to the adversary). By assuming a structure in the hypothesis space, we are able to exclude certain hypotheses, based on the results of each experiment. In the first Subsection we presented this model in a simple example on a totally ordered hypotheses space. There we observed the similarities between the induction and searching problems. Later, we expanded our analysis by reviewing a relatively recent work on sorting posets. We believe that, we may be able utilize the methodology in [20] for optimal searching and sorting in posets, to perform query optimal induction to a partially ordered hypotheses spaces. We give the following conjecture.

Conjecture 2 (Lower Bound: Queries in Partially Ordered Reasoning Framework)**.** *Assume a language $\mathcal{H}$ of hypotheses and $\mathcal{E}$ of the examples, and a formal language $\mathcal{L} \supseteq \mathcal{H} \cup \mathcal{E}$, and a pair of positive and negative examples $E^+, E^- \subseteq \mathcal{E}$, respectively. Consider the oracle function $\texttt{orac}\colon \mathcal{E} \to \{+, -\}$, that answers truthfully for an unkown adversary's hypothesis $h^\star$. Also, let $\mathcal{R} = \langle \mathcal{L}, \preceq \rangle$ be a* partially ordered *reasoning framework. There is no algorithm that finds $h^\star$ with less than $O(\log n + w)$ queries in the worst case, where w is the width of $\mathcal{R}$.*

1.6 Computational Learning Theory: PAC-learnability

In the previous Section we presented some results regarding the query complexity of exploring a structured space. Based on this result we stated some conjectures regarding lower bounds on the query complexity of inductive systems. The key argument of these conjectures states that based *only* on the structure of the hypothesis space we may derive lower bounds for the number of examples needed from a inductive system. Our conjectures assume non *a priori* knowledge of the distribution followed by the example space. In this Section we follow a different direction. We review an alternative existing model for analysing the query complexity of a learning algorithm (or inductive system), which is based on a probability theoretic analysis of both the hypothesis and the example space. We will see that, while following different directions we will arrive to similar logarithmic bounds.

The model we will introduce is the *Probably Approximate Correct learnability* or PAC-lernability [73]. A more gentle introduction in PAC-learnability can be found in [47]. Assume an example space $\mathcal{E}$. An adversary is modeled by an oracle function $\texttt{orac}\colon \mathcal{E} \to \{+, -\}$. As always $E^+ = \{e \in \mathcal{E} \mid \texttt{orac}(e) = +\}$ and $E^- = \{e \in \mathcal{E} \mid \texttt{orac}(e) = -\}$. Since the oracle function is *total* it holds that $E^- = \mathcal{E} \setminus E^+$. Additionally, the learning algorithm is agnostic about E^+, E^- and is only allowed to ask queries to the oracle. In this setting the learning algorithm is *not* allowed to ask about the membership of *any* example $e \in \mathcal{E}$. On the contrary it *randomly* chooses an example $e \in \mathcal{E}$ based on a distribution $\mathcal{D}$[17]. Also, consider a hypotheses space $\mathcal{H}$, and a hypothesis $h \in \mathcal{H}$. With $\text{error}_{\mathcal{D}}(h)$ we denote the *true error* of hypothesis h to *misclassify* an example $e \in \mathcal{E}$, drawn at random according to the distribution $\mathcal{D}$, with respect to an oracle function $\texttt{orac}(\cdot)$. We have,

$$\text{error}_{\mathcal{D}}(h) = \Pr_{e \sim \mathcal{D}}[\texttt{orac}(e) \neq h(e)]. \tag{1.14}$$

[17] In an ideal setting the distribution $\mathcal{D}$ encodes the possibility that a example e appears in nature. On the other hand, in a more realistic setting, it will encode the distribution followed by a given dateset. More often than not, the true distribution of a phenomenon differs from the distribution followed by the dataset. This fact is often referred to, by machine learning engineers, with the slogan *"a dataset is always biased"*.

Note that in equation (1.14) we used a different notation from previously in this Chapter. Here we regard a hypothesis as a *function approximator* for the oracle, i.e. $h\colon \mathcal{E} \to \{+,-\}$. This is a more general setting from the abstract reasoning frameworks defined above. In a poset setting the corresponding function approximator $\hat{h}$, for a hypothesis $h \in \mathcal{H}$, would be defined as $\hat{h}(e) = +$, if $h \succeq e$; $\hat{h}(e) = -$, otherwise.

Our aim is to characterize classes of target concepts that can be reliably learned from a *reasonable number of randomly drawn examples* and *a reasonable amount of computation*. In this direction, we define when a target concept, characterised by an oracle function $\texttt{orac}(\cdot)$, is Probably Approximately Correct. Consider a parameter $\varepsilon \in \left(0, \frac{1}{2}\right)$. In this setting we will say that a hypothesis $h \in \mathcal{H}$ is *approximately correct*, when $\text{error}_{\mathcal{D}}(h) \leq \varepsilon$. On the other hand, we demand a learner to find an approximate correct hypothesis $h \in \mathcal{H}$ with high probability. Thus, assume another parameter $\delta \in \left(0, \frac{1}{2}\right)$ and demand that,

$$\Pr_{h \sim \Delta}\left[\text{error}_{\mathcal{D}}(h) \leq \varepsilon\right] > (1-\delta), \tag{1.15}$$

where Δ is the induced distribution by the learner on the hypothesis space $\mathcal{H}$. Lastly, we want the learner to be able to find an approximate correct hypothesis in time

$$\text{poly}\left(\frac{1}{\varepsilon}, \frac{1}{\delta}, |\texttt{orac}|, \max_{e \in \mathcal{E}} |e|\right). \tag{1.16}$$

In the above equation with $|\texttt{orac}|$ we denote the *size of an encoding* of the adversary's oracle function $\texttt{orac}$. Additionally, in (1.16) we consider the *size of the encoding of the largest example*, as the last term. A concept, encoded by an oracle $\texttt{orac}(\cdot)$, is *PAC-learnable*, if there is a learner (or learning algorithm) that satisfies (1.15) in time given by (1.16).

Our definition of PAC-learnability may appear to be concerned only with the computational resources required for learning, whereas we are usually more concerned with the number of training examples required. However, the two are closely related. If a learner requires some minimum processing time per example, for a concept to be PAC-learnable by the learner, the latter *must learn from a polynomial number of training examples*. Additionally, the above definition of PAC-learnability *implicitly* assumes that the learner's hypothesis space $\mathcal{H}$ contains a hypothesis with *an arbitrary small error for every target concept*. Naturally this is difficult to assure if one does not know the oracle function in advance. Nevertheless, the results based on the PAC-learning model provide useful insights regarding the relative complexity of different learning problems and regarding the rate at which accuracy improves with additional training examples.

1.6.1 Simple Upper Bounds to Query Complexity

We now review a simple upper bound on the number of training examples for a *consistent* learner. A learner is consistent if it outputs hypotheses that fit *perfectly* the training data. We will derive an upper bound for this class of learners *interdependently* from the inner mechanics of the algorithm.

Assume a finite hypothesis space $\mathcal{H}$, with $|\mathcal{H}| < \infty$. Also, consider a sequence of identical and interdependently drawn (i.i.d.) set of examples $E \subseteq \mathcal{E}$, with respect to the distribution $\mathcal{D}$. The *version space* $V_{\mathcal{H},E} \subseteq \mathcal{H}$ includes all the hypotheses $h \in \mathcal{H}$ that correctly classify all the examples E, with respect to the target concept given by an oracle function $\texttt{orac}(\cdot)$. Formally,

$$V_{\mathcal{H},E} = \{h \in \mathcal{H} \mid \forall e \in E,\ \texttt{orac}(e) = h(e)\}. \tag{1.17}$$

We want to *bound* the probability that some hypothesis $h \in V_{\mathcal{H},E}$ is *approximate correct*. In this direction, we introduce the concept of an *ε-exhausted* version space. We say that a particular version space is ε-exhausted when *all* hypotheses h in $V_{\mathcal{H},E}$ are approximately correct, i.e. when,

$$\forall h \in V_{\mathcal{H},E},\ \text{error}_{\mathcal{D}}(h) < \varepsilon. \tag{1.18}$$

Note that when (1.18) holds, every hypothesis that is *consistent* with respect to the set of examples E is approximately correct.

Consider a hypothesis $h \in V_{\mathcal{H},E}$, which is not approximately correct. That is $\text{error}_{\mathcal{D}}(h) > \varepsilon$. Therefore the probability that h is consistent with a randomly drawn example would be $1 - \text{error}_{\mathcal{D}}(h) < 1 - \varepsilon$. Thus, the probability that h is consistent with all the examples from E is at most $(1-\varepsilon)^{|E|}$. Hence, $|V_{\mathcal{H},E}|(1-\varepsilon)^{|E|}$, and since $|V_{\mathcal{H},E}| \leq |\mathcal{H}|$, we have $|\mathcal{H}|(1-\varepsilon)^{|E|}$. Using the inequality $1 - x \leq e^{-x}$, for $x \in [0,1]$, where e is the Euler's number, we have proven the following theorem, due to Haussler.

Theorem 2 (Haussler, 1988 [47]). *Assume a finite hypothesis space $\mathcal{H}$ and a sequence of i.i.d. examples E, with respect to the distribution $\mathcal{D}$. Then for any $\varepsilon \in [0,1]$, the probability that the version space $V_{\mathcal{H},E}$ is* not *ε-exhausted is less than or equal to,*

$$|\mathcal{H}|e^{-\varepsilon|E|} \tag{1.19}$$

Above we proved an *upper bound* to the probability that the version space is *not* ε-exhausted, based on the number of training examples, the allowed error ε, and the size of $|\mathcal{H}|$. In other words, this bounds the probability that $|E|$, in number, training examples will *fail* to eliminate all the "bad" hypotheses, for any consistent learner using hypothesis space $\mathcal{H}$. In order to assure the PAC property we need to bound the probability that the learner will output a consistent but "faulty" hypothesis. Therefore, we demand $|\mathcal{H}|e^{-\varepsilon|E|} \leq \delta$. Solving for $|E|$ we have,

$$|E| \geq \frac{1}{\varepsilon}\left(\ln|\mathcal{H}| + \ln\left(\frac{1}{\delta}\right)\right), \tag{1.20}$$

where $\ln(\cdot)$ is the Napierian logarithm.

Note that equation (1.20) is an *upper bound* on the number of examples needed by a consistent learning algorithm. That is, there *always* exists an algorithm that provided this many examples will PAC-learn any given target concept, i.e. it will respect (1.15). Such an algorithm is the *trivial* one, where the learner will sequentially traverse all hypotheses of $\mathcal{H}$ testing whether the current hypothesis $h \in \mathcal{H}$ is consistent with the example set E. Note that the trivial algorithm will make, *at most*,

$$|\mathcal{H}| \cdot \frac{1}{\varepsilon}\left(\ln|\mathcal{H}| + \ln\left(\frac{1}{\delta}\right)\right) \cdot t_{\text{proc}} \tag{1.21}$$

time, where t_{proc} is the processing time for each example. The trivial algorithm will be *polynomial* (i.e. satisfies (1.16)) only when $|\mathcal{H}| = \text{poly}(|\texttt{orac}|)$ and $t_{\text{proc}} = \text{poly}(|e_{\max}|)$, where $e_{\max}$ is the example with the lengthier encoding.

In this Section we discussed briefly some elementary results from *Computationally Learning Theory* and more precisely the PAC model. In (1.20) we proved a simple upper bound to the *sample complexity* of a consistent learning algorithm. Note that in this model the learner is only allowed to ask the adversary only abound randomly drawn examples according to an unknown distribution $\mathcal{D}$. There are related results in this area that lift the consistency hypothesis, namely where a hypothesis with an arbitrary small error ε cannot be found. There again the number of examples needed is logarithmic to the size of the hypothesis space. A computational model which lifts the finite hypothesis space assumption is the *Vapnik-Chervonenkis Dimension* or VC-Dimension. The interested reader is referred to [47] for a more in depth discussion about Computational Learning Theory.

1.7 Conclusions

In this introductory Chapter we review some peripheral related notions for this paper. Firstly we reviewed a well established model for measuring the information of a structure. Then we discussed a well behaved structure for the hypothesis space, that is partial orders and lattices. We introduced the concept of an *Abstract Reasoning Framework*, we also defined the key problems related to reasoning in partially ordered hypothesis spaces. Additionally, we introduced the concept of *Query-Based* induction through examples. We provided some conjectures that we believe provide a fruitful area for future research. In this direction, we presented an interesting methodology from Daskalakis et al. that may be utilised for the induction problems. The authors of the latter work were able to achieve optimality for sorting and searching problems in partially ordered sets, thus closing the gap between the information-theoretic lower bound and the corresponding upper bound. Lastly, we discussed an alternative model for proving lower bound for inductive systems that has become the standard in the machine learning community. The latter model differs from our perspective in assuming a distribution over the example space, prohibiting the inductive system from asking the adversary arbitrary queries, while not taking into advantage the structure of the hypothesis space.

In the next Chapter we will review deductive reasoning frameworks. From now on, all the hypothesis language we will focus on will be subsets of mathematical logic. Later, we will present induction on logical languages, where the abstract traversal operators introduced here will become precise.

Chapter 2

Classical Logic

An intelligent system is required to have two main capabilities for reasoning; it should be able to form conclusions from previous knowledge, and learn new knowledge from past experience. The first way of thinking moves an intelligent agent from the general to specific, whereas the latter move from specific to general. The former reasoning mechanism is known as *deduction*, while the latter is known as *induction*. An important achievement of the AI researchers of the past century is that they manage to unify these to elementary reasoning functionalities to a single theory [8, 61]. This is constitutes a significant result of symbolic AI and highlights the benefits of a logic based approach to artificial intelligence.

In this chapter we discuss deductive reasoning. Our goal is to establish the foundations on which we will based when discussing inductive reasoning in the next chapter. We introduce some elementary concepts from classical mathematical logic[1]. We focus in two traditional logical languages, namely propositional and first-order logic. The concepts introduced here can be found in any textbook on mathematical logic, e.g. [23]. Mathematical logic has numerous applications in artificial intelligence, many of which are discussed in [33].

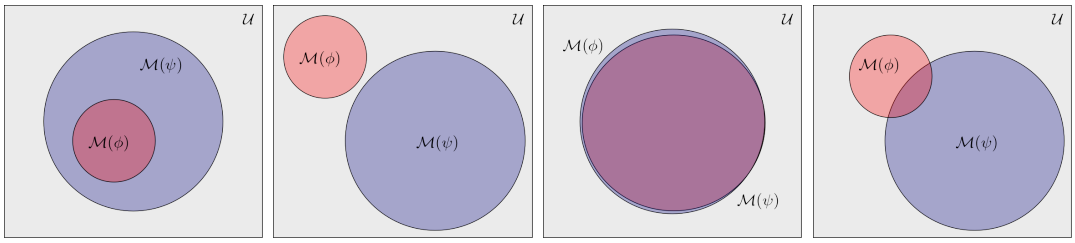
2.1 Syntax of Mathematical Logic

The two elementary formal languages of mathematical logic is *propositional logic* (PL) and *first order logic* (FOL). In both languages we use the connectives $\wedge, \vee, \neg, \rightarrow$ and parentheses $(,)$. In PL we use an enumerable infinite set of propositional variables V. A formula is a finite sequence of propositional variables combined by the above connectives. An interpretation u in PL is an assignment from V in $\mathbb{B} = \{0, 1\}$, namely $u \colon V \rightarrow \mathbb{B}$.

On the other hand, in FOL we can define a variety of languages. Again, we have an infinite set of variables V. Each language in FOL is defined by a *signature* $\tau = \langle C, F, P \rangle$, where C denotes a set of *constants*, F consists of *function* symbols and P is a set of *predicates*. Each predicate of P is associated with an *arity*, i.e. the number of arguments it takes. Also, each element in F is associated with an arity. A function can take as arguments other functions, constants, or variables. A *term* is a function with each of its arguments fixed. A predicate may take as arguments, functions, constants, or variables. An *atom* is a predicate with all its arguments fixed. In FOL a formula is built from a set of atoms using the logical connectives and the quantifiers $\forall, \exists$. For a languages with signature $\tau = \langle C, F, P \rangle$, and a non-empty domain D, an interpretation is a mapping from C to elements of D, from functions F to functions on D and from the set of predicates P to relations on D.

In this section we discuss some elementary notions of mathematical logic. We assume an abstract language $\mathcal{L}$ which is either PL or one of the FOL languages.

[1] We emphasise that here we review ideas from "classical" mathematical logic, to distinguish between more modern approaches to formal reasoning, such as non-monotonic logics, modal logics, temporal logics etc. Additionally, we also distinguish between classical and clausal logic, presented in the next chapter. Despite that clausal logic does not constitute a new formalism, since it is a subset of classical logic, from historical perspective it became popular much latter, in the '70s, with the advent of logic programming.



(a) $\phi \models \psi$ **(b)** $\phi \models \neg\psi$ **(c)** $\phi \models\!\models \psi$ **(d)** $\phi \not\models \psi$

Figure 2.1: All the potential relations between two formulas $\phi, \psi \in \mathcal{L}$ and their corresponding sets of models $\mathcal{M}(\phi), \mathcal{M}(\psi)$. With $\mathcal{U}$ we denote the set of all models.

2.2 Semantic Entailment

Let $\phi \in \mathcal{L}$ be a formula of a formal language $\mathcal{L}$. We say that an interpretation u is a *model* of ϕ if it satisfies ϕ, and we write $u \models \phi$. We denote with $\mathcal{M}(\phi)$ the set of all models of ϕ. Let $\phi, \psi \in \mathcal{L}$ be two formulas of a formal language $\mathcal{L}$. We say that ϕ *entails* ψ, when *every model* of ϕ also models ψ. We denote this fact with,

$$\phi \models \psi. \tag{2.1}$$

From the above definition of semantic entailment we have that,

$$\phi \models \psi \Leftrightarrow \mathcal{M}(\phi) \subseteq \mathcal{M}(\psi). \tag{2.2}$$

When ϕ does not entail ψ we write $\phi \not\models \psi$. Note that $\phi \not\models \psi$ is not equivalent to $\phi \models \neg\psi$. When $\phi \models \neg\psi$ we have $\mathcal{M}(\phi) \subseteq \overline{\mathcal{M}(\psi)}$. When $\phi \not\models \psi$, in general we could have both $\mathcal{M}(\phi) \cap \mathcal{M}(\psi) \neq \varnothing$ and $\mathcal{M}(\phi) \cap \overline{\mathcal{M}(\psi)} \neq \varnothing$. Lastly, when $\mathcal{M}(\phi) = \mathcal{M}(\psi)$ we write $\phi \models\!\models \psi$, and say that ϕ, ψ are logically (or semantically) *equivalent*. In Figure 2.1 we depict all the possible relations between two formulas ϕ, ψ and their corresponding set of models.

Let $\mathcal{U}$ be the set of all models of the formal language $\mathcal{L}$. When $\mathcal{U} \models \phi$, namely *every model* satisfies the formula ϕ, we omit the set of models and write $\models \phi$. We call a formulas that is satisfied by every model a *tautology*. When $\models \neg\phi$, we say that ϕ is a *contradiction*. If ϕ is a tautology, then $\neg\phi$ is a contradiction and vice versa.

2.3 Syntactic Entailment

Another important relation between formulas in a formal language $\mathcal{L}$ is *syntactic* entailment. Let $T \subseteq \mathcal{L}$ be a set of formulas. Also, consider the deductive rule,

$$\frac{\phi \qquad \phi \rightarrow \psi}{\psi} \tag{2.3}$$

The deductive rule of (2.3) is called *modus ponens*. We write $T \vdash \phi$ when there is a finite sequence $\chi_1, \chi_2, \ldots, \chi_n = \phi$, where for each χ_i, either $\chi_i \in T$, or there is some χ_k, χ_l, with $k, l < i$ and χ_i is the result of modus ponens applied on χ_k, χ_l[2]. The sequence $\chi_1, \chi_2, \ldots, \chi_n$ is called a *formal proof*. When $T \vdash \phi$ we say that T *proves* ϕ.

Lastly, we say that some set of formulas $T \subseteq \mathcal{L}$ is *consistent*, when there is no formula $\phi \in \mathcal{L}$ such that,

$$T \vdash \phi \text{ and } T \vdash \neg\phi. \tag{2.4}$$

Namely, a set of formulas T does not prove ϕ and its negation.

[2]Modus ponens is the main deductive rule we are going to use in the sequel of this paper. For sake of completeness we mention that in FOL we use two additional deductive rules, namely *generalization* $\phi \Rightarrow \forall v\phi$ and *existence elimination* $\phi \rightarrow \psi \Rightarrow \exists v(\phi \rightarrow \psi)$.

2.4 Soundness & Completeness

Thus far we have come up with two different notion for derivation, semantic and syntactic. In this paragraph we show that these two methods coincide, providing two alternative methods for deriving the truth of a statement. In order to ensure that, we need to establish the validity of two properties, *soundness* and *completeness*.

For a formal language $\mathcal{L}$, and a set of formulas $T \subseteq \mathcal{L}$, the soundness property ensures that,

$$T \vdash \phi \Rightarrow T \models \phi. \tag{2.5}$$

Both PL and FOL are sound [23].

The inverse property, namely,

$$T \models \phi \Rightarrow T \vdash_{\mathcal{A}} \phi, \tag{2.6}$$

is called completeness. Both PL and FOL are complete [23]. Note the subscript in the equation 2.6, that is because in order to ensure completeness we need to provide a set of axioms $\mathcal{A}$. Such an established set of axioms are Hilbert's axioms [23]. In other words, when we write $T \vdash_{\mathcal{A}} \phi$, we mean $T \cup \mathcal{A} \vdash \phi$. Note that the set of Hilbert's axioms $\mathcal{A}$ is enumerable infinite, thus constituting an *axiomatic scheme*, i.e. a infinite set of axioms that admits a finite description. Intuitively a set of axioms describes a set of *background knowledge*, of basic truths we need to include in our deductive system to ensure completeness.

2.5 Abstract Reasoning Frameworks in Classical Logic

The inference operators we introduced in this chapter can be used to define abstract reasoning frameworks following Definition 5. Note that both *semantic* entailment $\models$ and *syntactic* entailment $\vdash$ induce a *quasi* order on a formal language $\mathcal{L} \in \{\mathsf{PL}, \mathsf{FOL}\}$. Note that the existence of *syntactic variants* prohibit the structures $\langle \mathcal{L}, \models \rangle$ and $\langle \mathcal{L}, \vdash \rangle$ from being *partial orders*. For example consider the formulas $\neg\neg\phi$ and ϕ. Then $\neg\neg\phi \models\!\!\!\models \phi$, but $\neg\neg\phi \not\equiv \phi$. Thus, in order to introduce a partial order we need to partition $\mathcal{L}$ with respect to the logical equivalence $\models\!\!\!\models$. Naturally, this introduces an additional, *not negligible* in general, computational cost. In the sequel, we will often *implicitly* assume a partially ordered space, with respect to the equivalence relation $\models\!\!\!\models$. In this direction, we introduce two abstract reasoning frameworks, based on the above observations.

Definition 6 ($\mathcal{L}$-Semantic Abstract Reasoning Framework)**.** *Assume a formal language $\mathcal{L} \in \{PL, FOL\}$. Also consider the* semantic entailment *relation $\models$. We call the structure $\langle \mathcal{L}, \models \rangle$ the* $\mathcal{L}$-Semantic *abstract reasoning framework. Additionally, assume the partition on $\mathcal{L}$ with respect to the semantic equivalence $\models\!\!\!\models$. We will denote the partitioned set $[\mathcal{L}]$, while we call the corresponding structure the* $[\mathcal{L}]$-Semantic *reasoning framework.*

Following the above definition we distinguish between the PL-Semantic and FOL-Semantic reasoning framework. In Figure 2.2 we present the hypothesis space for the $\{a, b, c\}$-Semantic reasoning framework. In this example we consider a very limited language constituted only from three propositional variables and the connectives $\wedge, \vee$. Observe that in this case the hypothesis space is a *lattice*. In Figure 2.2 becomes apparent the choice for the notation for the join (eq. 1.3) and meet (eq. 1.4) operations on lattices. *The lattice symbols coincide with their logical counterparts.* This correspondence is of key importance in our analysis, since it translates lattice operations (or traversal operations) into logical operations and vice versa. For example consider the formulas $a \wedge (b \vee c)$ and $b \wedge (a \vee c)$. Consider their lower upper bound, namely $[a \wedge (b \vee c)] \wedge [b \wedge (a \vee c)]$. This upper bound coincides with the smallest (in length) formula that satisfies their logical conjunction, i.e. $(a \wedge (b \vee c)) \wedge (b \wedge (a \vee c))$. This formula is $a \wedge b$.

Symmetrically, we define the $\mathcal{L}$-Syntactic reasoning frameworks.

Definition 7 ($\mathcal{L}$-Syntactic Abstract Reasoning Framework)**.** *Assume a formal language $\mathcal{L} \in \{PL, FOL\}$. Also consider the* syntactic entailment *relation $\vdash$. We call the structure $\langle \mathcal{L}, \vdash \rangle$ the* $\mathcal{L}$-Syntactic *abstract reasoning framework. Additionally, assume the partition on $\mathcal{L}$ with respect to the semantic equivalence $\models\!\!\!\models$. We will denote the partitioned set $[\mathcal{L}]$, while we call the corresponding structure the* $[\mathcal{L}]$-Syntactic *reasoning framework.*

Again, we distinguish between the the PL-Syntactic and FOL-Syntactic reasoning frameworks. It is important to note that both $\mathcal{L}$-Semantic and $\mathcal{L}$-Syntactic reasoning frameworks

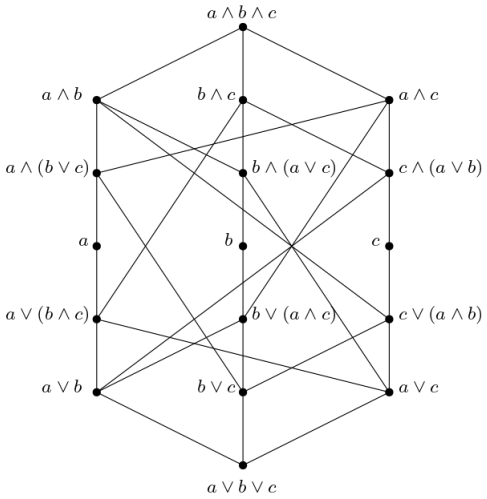


Figure 2.2: The hypothesis space for $\{a, b, c\}$-Semantic when we only use $\wedge, \vee$ as connectives, in a propositional context.

are *infinite*. Note that the *soundness* and *completeness* properties can be express as simple set theoretic relations in the reasoning framework language. Namely, for soundness we have,

$$\mathcal{L}\text{-Syntactic} \subseteq \mathcal{L}\text{-Semantic}, \tag{2.7}$$

while the completeness property can be expressed as,

$$\mathcal{L}\text{-Semantic} \subseteq \mathcal{L}\text{-Syntactic}. \tag{2.8}$$

In the above equation we mainly refer to set-theoretic inclusion between the *relations*. Thus, we may regard the syntactic entailment $\vdash$ as an *approximation* of the semantic entailment $\models$. In this case, of course, the approximation is perfect. In the following chapter we will present various approximations to $\models$. Observe that given a reasoning framework $\mathcal{R} = \langle \mathcal{L}, \preceq \rangle$ we can obtain an approximation $\mathcal{R}'$, by either taking a subset of the formal language $\mathcal{L}' \subseteq \mathcal{L}$, or a subset of the generalisation relation $\preceq' \subseteq \preceq$ (or, of course, both). In the following chapter we will define both subsets for $\mathcal{L}$ and subsets for $\preceq$. Before we close this section we discuss reasoning frameworks on *sets* of formulas.

2.5.1 Reasoning Framework on Sets

In the above notation we make a subtle assumption regarding the syntax of a logical formula. Note that in Section 2.3 we defined syntactic entailment using modus ponens (see Equation 2.3). There we defined modus ponens on a *sequence* of logical formulas. On the other hand, in Definition 7 we approached semantic entailment on single formulas. These two approaches are in fact equivalent since a sequence of formulas is logically equivalent to their conjunction. Thus a sequence of formulas can be treated as a single long formula. Despite that inductive systems can be separated between those handling sets of formulas and those handling a single formula. Additionally, there are some subtleties when dealing with sets of formulas. Thus, we define the following reasoning framework on sets.

Definition 8 ($2^{\mathcal{L}}$-Syntactic Abstract Reasoning Framework)**.** *Assume a formal language $\mathcal{L} \in \{PL, FOL\}$, and the family of sets of formulas in this language $2^{\mathcal{L}}$. Also consider the* syntactic entailment *relation $\vdash$. We call the structure $\langle 2^{\mathcal{L}}, \vdash \rangle$ the $2^{\mathcal{L}}$-Syntactic abstract reasoning framework. Additionally, assume the partition on $2^{\mathcal{L}}$ with respect to the semantic equivalence $\models\!\models$. We will denote the partitioned set $[2^{\mathcal{L}}]$, while we call the corresponding structure the $[2^{\mathcal{L}}]$-Syntactic reasoning framework.*

Observe that in order to traverse $\mathcal{L}$-Syntax we will use generalization and specialisation operators such as those of equations (1.7) and (1.8). On the other hand, in order to traverse $2^{\mathcal{L}}$-Syntactic we will be using the operators (1.9) and (1.10).

2.6 Conclusions

In this chapter we discussed mathematical logic. We examined the two foundational formalisms, namely propositional logic (PL) and first-order logic (FOL). We presented the syntax of the two formalisms in Section 2.1. In Section 2.2 we presented semantic entailment, while in Section 2.3 we reviewed it syntactic counterpart. These concepts provide us with a formal way of deductive reasoning. In Section 2.4 we established the equivalence of these operators, in both propositional and first-order logic. Lastly, in Section 2.5 we discussed the underlying structure that appears in these logical language in a form of a lattice. We close this chapter with some notes on the computational properties of classical logic and its impact to artificial intelligence.

2.6.1 Notes on Computational Properties of Classical Logic

The completeness of FOL has been proved from Gödel in 1929, and provided a robust argument for the use of mathematical logic in artificial intelligence [48]. That is because, it ensures that we can explore the the space of truth sentences syntactically. Gödel's completeness theorem set the foundations for *automated theorem proving*. In 1931 Gödel publishes his *incompleteness* result. It states that any first order language L strong enough to describe basic arithmetic, i.e. models the Peano's arithmetic, is not complete. In other words, there is some formula ϕ, that is true but cannot be proved formally. The problem with incompleteness lays in the fact that Peano's axioms admits models alternative to traditional arithmetic. We can still arrive to truths that lay in the intersection of all models of arithmetic, but there are arithmetic truths that lay beyond this intersection. The issue of having unwanted models of a formal description concerns also the AI community, such an example would be the *frame problem* [48, 33].

We conclude the discussion on Classical Logic with some remarks on the computational properties of PL and FOL. Firstly observe that PL $\subseteq$ FOL, indeed we can encode any language of PL as a FOL language, we just use predicates of arity 0. On the other hand, every formula ϕ in PL has a finite set of possible models, i.e. 2^n, where n is the number of variables appear in ϕ. On the other hand, a formula ϕ of FOL using at least a predicate of non-zero arity has a *infinite* set of possible models. Thus, the SATIFIABILITY problem in PL is *decidable*, while the same problem in FOL is *undecidable* [67]. Note that the SATIFIABILITY problem is a well known NP-hard problem. The above concerns lead us to consider alternative formalisms for automated reasoning, in order to avoid some of the computational expense. In the next section, we discuss such formalisms.

Chapter 3

Clausal Logic

In this chapter we introduce some elementary notions in order to ease automated reasoning in classical logic. The results presented here focus on a particular subset of classical logic that regards conjunction normal forms. This notations enables us to syntactically manipulate clauses easier. Clausal logic lays at the core of automated theorem proving and logic programming [33]. In this chapter we follow the presentation of [61]. All the results presented here can also be found to related text-books such as [33].

Let $\phi \in \mathcal{L}$ be a formula of a formal language $\mathcal{L}$. We can always write ϕ as conjunction of disjunctions, namely *"ands of ors"* [34], when ϕ is written in such form we say it is in *conjunctive normal form* (CNF). A *literal* is an atom or its negation. A *clause* is a disjunction of literals. A clause in *set notation* is written as $\{A_1, A_2, \ldots, A_n, \neg A_{n+1}, \ldots \neg A_{n+m}\}$. For a formula $\phi \in \mathcal{L}$ in CNF, we denote with $\mathsf{Claus}(\phi)$ the set of clauses in ϕ. Alternatively, we write a clause as,

$$A_1; A_2; \ldots; A_n \leftarrow A_{n+1}, \ldots, A_{n+m}. \tag{3.1}$$

In the above notation ; stands for $\vee$ and , stands for $\wedge$. For the clause of (3.1), we say that the atoms before $\leftarrow$ constitute the *head*, while the atoms after $\leftarrow$ compose the *body* of the clause. In FOL we assume that each variable of each atom is *universally quantified*. When $n = 1$ and $m = 0$ a clause is called a *fact*. For a *definite clause* we have $n = 1$. A *Horn clause* has $n = 1$ or $n = 0$. lastly, a *denial* is a clause where $n = 0$. A *substitution* is a mapping θ from variables to terms. A *ground term* is a term that does not include a variable. A ground atom is an atom which each of its arguments is a ground term.

3.1 Semantics of Clausal Logic

In order to automate deduction in FOL we need to constraint the search space of possible models of a set of clauses. Thus, we introduce the notion of *Herbrand universe*. The Herbrand universe (or domain) is defined as the set of *all ground terms* that can be constructed using the constants and function symbols that occur in the set of clauses considered. Now a *Herbrand interpretation* maps each ground term to itself. In similar manner, predicates are mapped onto relations over the Herbrand universe. The result is that a Herbrand interpretation can be viewed as the *set of ground atoms that are true in the interpretation*. The atoms not included in the interpretation are assumed to be *false*.

Let C be a set of classes and u a Herbrand interpretation. We say that u *models* C, and write $u \models C$, if and only if for each clause $c \in C$, with $c \equiv A_1; A_2; \ldots; A_n \leftarrow A_{n+1}, \ldots, A_{n+m}$ and all *ground substitutions* θ,

$$\{A_{n+1}\theta, \ldots, A_{n+m}\theta\} \subseteq u \Rightarrow \{A_1, A_2, \ldots, A_n\} \cap u \neq \varnothing. \tag{3.2}$$

The following theorem will allows us to focus out attention on Herbrand interpretations and models to reason about the satisfiability of a particular formula.

Theorem 3 (Nienhuys-Chen and de Wolf, 1997 [61])**.** *Let C be a set of clauses. The set of clauses C has a model* if and only if *has a Herbrand model.*

Let $\leftarrow$ be the empty clause, with neither a head, not a body, sometimes written as $\square$. Form Theorem 3 and the definition of Herbrand interpretations in (3.2) note that $\square$ is a *contradiction*, since the body of the clause is always included to any interpretation, and the body cannot have an intersection with any interpretation. The following theorem allows as to utilize Herbrand models in order to check for entailment.

Theorem 4 (Nienhuys-Chen and de Wolf, 1997 [61])**.** *Let C be a set of clauses and c be a clause. Then $C \models c$ if and only if $C \wedge \neg c \models \square$.*

In general, a theory (or set of clauses) has *multiple models*. Additionally, note that the models form a partial order, with respect to set inclusion relation. For example consider the clause, of PL, $c \equiv a \leftarrow b$. Then $u = \varnothing$ is a model of c. On the other hand, $u' = \{a, b\}$ is also a model. Naturally, $c \subseteq c'$. This motivates us to introduce the notion of *minimal* model. A Herbrand model is called minimal when none of its subsets is also a model. In *definite* clauses, namely when there is only a single atom on the head, the minimal model is *unique*. Thus, it is called the *least Herbrand model*. The least Herbrand model is an important concept, since it captures the semantics of a set of clauses. It consists of *all ground atoms that are logically entailed by the definite clause theory.*

3.2 Inference in Clausal Logic

In clausal logic the main inference rule is known as *resolution*. The resolution principle was introduced by Robinson in 1965 [61]. Assume the clauses in PL $c_1 = \{a, \neg b, \neg c\}$ and $c_2 = \{d, \neg e, \neg a\}$. Observe that we can eliminate a from c_1 and $\neg a$ from c_2, resulting in $c_3 = \{d, \neg e, \neg b, \neg c\}$. We call the clause c_3 the *resolvent* of c_1, c_2. In this paper we restrict our attention to resolution for Horn clauses, remember that a Horn clause has only one positive literal. The resolution inference rule for propositional Horn clauses is presented in relation (3.3).

$$\frac{\begin{array}{l} h \leftarrow b_1, \ldots, b_n \\ g \leftarrow c_1, \ldots, c_{i-1}, h, c_i, \ldots, c_m \end{array}}{g \leftarrow c_1, \ldots, c_{i-1}, b_1, \ldots, b_n, c_i, \ldots, c_m} \tag{3.3}$$

Using resolution we can construct proofs as we did with modus ponens. The resolution operator is sound, but not complete [61]. On the other hand, resolution is *refutation complete*, meaning that when $C \models \square$, then $C \vdash_{\text{res}} \square$.

Theorem 5 (Nienhuys-Cheng and de Wolf, 1997 [61])**.** *Let C be a set of clause. Then C is* unsatisfiable*, that is $C \models \square$, if and only if there exists a resolution derivation of the empty clause $\square$ starting from C.*

Observe the difference between completeness and refutation completeness. Assume the facts $c_1 = \{a\}$ and $c_2 = \{b\}$. Then, $c_1, c_2 \models a \wedge b$, but we are unable to arrive to $a \wedge b$ using resolution. Using Theorems 4, 5 and resolution's soundness we can compose a procedure for deciding the entailment of a logical formula. Deciding whether a set of Horn clauses logically entails a Horn clause, i.e. $C \models h \leftarrow b_1, \ldots, b_n$, can be realized a follows:

1. Negate the clause $c \equiv h \leftarrow b_1, \ldots, b_n$ to $c' = \{\neg h, b_1, \ldots, b_n\}$
2. Try to derive the empty clause $\square$ from $C \wedge c'$, that is $C \wedge c' \vdash \square$.

So far we introduced the mechanics of resolution in propositional Horn clauses. In order to transfer our method to first order Horn clauses, we need to manipulate variables. In this direction we present *unification*. Unification is needed to make a literal in one clause match a literal in the other clause. For example, assume the clause $c_1 \equiv$ `father(X,Y)` $\leftarrow$ `parent(X,Y),male(X)`. We want to resolve c_1 against $c_2 \equiv \leftarrow$ `father(anakin,luke)`. It is necessary to unify the literals `father(X,Y)` and `father(anakin,luke)` using the substitution $\theta =$ {`X/anakin,Y/luke`} to yield the clause $\leftarrow$ `parent(anakin,luke),male(anakin)`. Formally a *unifier* of two atoms f_1, f_2 is a substitution such that $f_1\theta \equiv f_2\theta$.

Notice also that unifiers do not always exist; for instance, the atoms `father(john,frank)` and `father(X,paul)` do not unify. To guarantee the refutation completeness of resolution when working with first order definite clauses it is necessary to work with a special type of unifier, which is called the *most general unifier*. With $\theta\rho$ we denote the substitution that first applies θ and then ρ.

Definition 9. *Let F_1, F_2 be two formulae and θ a unifier for them. Then θ is called* the most general unifier *if for every other unifier σ of F_1, F_2, we have $\sigma = \theta\rho$, where ρ is a non trivial substitution. The trivial substitution is the* identity *substitution.*

3.3 Logic Programming

Now that we established the core notions when dealing with clausal logic we are ready to describe its application to Logic Programming and specifically to the programming language Prolog. Prolog was first developed and implemented in Marseille, France in 1972 by Alain Colmerauer with Philippe Roussel, based on Robert Kowalski's procedural interpretation of Horn clauses at University of Edinburgh [38]. The name "Prolog" is an abbreviation of *"programmation et logique"*, i.e. "programming and logic". Prolog is essentially a automated theorem prover for Horn clauses. A Prolog programme is a set of Horn clauses. When the user compiles her programme P, in a compiler like SWI-Prolog [75] or ECliPSe Prolog [64], she will be greeted with a prompt requiring a query q. The query q is a disjunction of atoms. Then, the Prolog compiler attempts to construct the derivation $P \vdash q$. In order to achieve that, Prolog follows the procedure described in the previous subsection, namely negates q, and attempts to prove whether $P \wedge q \vdash \square$ using resolution. When the compiler is unable to do that it returns `false`.

Prolog uses a variation of resolution, known as *SLD-resolution*. The abbreviation SLD stands for *Selective Linear Definite clause resolution*, which describes a strategy when searching the atoms of a clause for unification. SLD-resolution chooses always the leftmost atom for unification. Thus, implementing a *depth-first* search strategy. Note that this strategy is *incomplete* in the sense that the derivation procedure may stack on an infinite loop, and fail to prove a query, even when such a derivation is possible using resolution. Thus, in contrast to mathematical logic where the and "," operator is communicative, the syntactic order of the atoms in a query q matters in Prolog. A complete strategy could be achieved if Prolog used a *breadth first* search strategy. On the other hand, such a strategy comes with a forbidding memory expense [61].

Computational Aspects. Prolog allows for recursion, meaning that the atom on the head of a clause can be repeated in its body. This property makes Prolog *Turing-complete*, meaning that any programme can be expressed in Prolog. The SATISFIABILITY problem for a set of Horn clauses in PL can be solved in *polynomial* time [67]. On the other hand, the same problem for *first-order* Horn clauses is in NP-hard [33], which should come as no surprise since Prolog is Turing complete. An interesting survey regarding the expressive power or Prolog and its complexity can be found at [55, 29]. Horn clauses, and the programming language Prolog achieves a well balance between expressivity and complexity. A similar family of languages known as *Datalog* restricts the use of recursion in order to limit the computational expenses [14].

Closed World Assumption (CWA). An important assumption in the most implementations of Prolog is the *closed world assumption*, introduced by the operator `not`. The `not` operator treats *negation as failure*. The CWA can be summarised in the following equation,

$$C \nvdash q \Rightarrow C \vdash \texttt{not}\ q. \tag{3.4}$$

Note, that `not` fundamentally differs from the logical negation $\neg$, as we discussed in the previous subsection, where we highlighted that $C \not\models q \not\Rightarrow C \models \neg q$. The utilisation of negation as failure, pushes the Prolog formalisation into the realm of *non-monotonic* logic. Let $T \subseteq \mathcal{L}$ be a set of formulas of a formal language $\mathcal{L}$, with $\mathsf{Cons}(T)$ we denote the set of formulas entailed from T, i.e. $\mathsf{Cons}(T) = \{\phi \in \mathcal{L} \mid T \models \phi\}$. Classical logic is *monotone* in the sense that if $T \subseteq T'$, then $\mathsf{Cons}(T) \subseteq \mathsf{Cons}(T')$. In other words, when we obtain more knowledge, non of our previous beliefs (or *conclusions*) changes. On the other hand, in a non-monotonic setting some of our beliefs may become invalid when our knowledge expands. For example, consider the Prolog programme $p \equiv a \leftarrow \texttt{not}\ b$. Thus we have $p \vdash a$. On the other hand, $p \wedge b \nvdash a$. This characteristic of logic programming makes it more suitable as formalism for *"common-sense reasoning"* and artificial intelligence.

3.3.1 Answer Set Programming

Another reasoning paradigm, similar to logic programming, is *Answer Set Programming (ASP)*. Again, an ASP programme is just a set of clauses. Contrary to Prolog, a query clause q is not required in ASP. The product of computation (reasoning) in ASP is a *minimal model*, known as *answer set* or *stable set* [33]. Additionally, ASP allows for various extensions of the traditional Prolog syntax, including *aggregators*, *choice rules*, and *constraints*. The inference method used by an ASP reasoner is again SLD-resolution with negation as failure[1]. Despite these additions, ASP implementations, like Clingo [28], allow only for bounded recursion, thus keeping the language from being Turing complete [12][2]. On the other hand, due to this restriction, ASP programmes almost always terminate [6]. Lastly, because ASP computes a complete model, rather than answering a query, its inference usually comes with a great computationally cost. For all the above reasons ASP is best suited as solver for complex constraint satisfaction problems, rather than a complete programming paradigm.

3.4 Abstract Reasoning Frameworks in Clausal Logic

In this section we will see our first approximation for the reasoning frameworks introduced in Chapter 2. Our approximation will be mainly with respect to $\mathcal{L}$-Syntactic. Consider the languages $\text{Horn}_{\mathsf{PL}}$ and $\text{Horn}_{\mathsf{FOL}}$ corresponding to propositional and first order Horn clauses respectively. Naturally, the Horn refinements are subsets of the corresponding full versions, i.e. $\text{Horn}_{\mathsf{PL}} \subset \mathsf{PL}$ and $\text{Horn}_{\mathsf{FOL}} \subset \mathsf{FOL}$. Additionally, we consider a refinement of syntactic entailment, namely resolution, denoted with $\vdash_{\text{res}}$. From the previous discussion it should be obvious that $\vdash_{\text{res}} \subset \vdash$, while the inverse does *not* hold. Note that $\vdash_{\text{res}}$ is a *binary* specialisation operator. Namely, it operates on a set of at least two clauses (see Equation 3.3). Therefore, the elements abstract reasoning framework considered here will be *sets* of clauses. Assume the set $\text{Horn} = \{\text{Horn}_{\mathsf{PL}}, \text{Horn}_{\mathsf{FOL}}\}$, we will consider the reasoning framework 2^{Horn}-Resolution which we define as follows.

Definition 10 (2^{Horn}-Resolution Abstract Reasoning Framework)**.** *Assume a formal language* $\text{Horn} \in \{\text{Horn}_{PL}, \text{Horn}_{FOL}\}$*, and the family of sets of formulas in this language* 2^{Horn}*. Also consider the* resolution *relation* $\vdash_{res}$*. We call the structure* $\langle 2^{\text{Horn}}, \vdash_{res} \rangle$ *the* 2^{Horn}-Resolution *abstract reasoning framework. Additionally, assume the partition on* 2^{Horn} *with respect to the semantic equivalence* $\models\models$*. We will denote the partitioned set* $[2^{\text{Horn}}]$*, while we call the corresponding structure the* $[2^{\text{Horn}}]$-Resolution *reasoning framework.*

Usually, in order to ease notation we will be referring to $[2^{\text{Horn}}]$-Resolution just as 2^{Horn}-Resolution. We will revisit the reasoning framework of Definition 10 in Chapter 5, where we inverse the resolution operator, thus performing induction. Note that in practice is often useful to consider a *finite* subset of the Horn language in order to keep the hypothesis space finite, and its exploration tractable. This technique is known as *language bias* and we will be discussing it in Chapter 6. For example consider a definite programme P. Naturally, every programme is finite. The reasoning framework Prolog's reasoner traverses will be 2^P-Resolution. In Figure 3.1 we present a simple example of the reasoning framework induced by the definite programme $P = \{a \leftarrow, b \leftarrow, c \leftarrow, d \leftarrow a, b, c\}$. Note that if $P \subset \text{Horn}_{\mathsf{PL}}$, with P being a finite propositional definite programme, then the reasoning framework will also be finite. On the other hand, this does not hold for a first order programme, i.e. $P \subset \text{Horn}_{\mathsf{FOL}}$, where we use at least one functional symbol, that is not a constant. In the latter case the Herbrand universe will be infinite.

Note that in Figure 3.1 we denote with the red edges the route that will be followed by Prolog's SLD-Resolution. Additionally, observe the exponential growth of the hypothesis space, even in such small examples. This is the reason why Prolog implements a DFS traversal instead of BFS. Even in the above simple example, we may observe the exponential expansion of BFS' queue.

In order to traverse reasoning frameworks constituted of sets of clauses we use the operators (1.9) and (1.10) introduced in Chapter 1. In Figure 3.1 we discarded the sets induced by elimination of clauses in order to ease the presentation.

[1] Sometimes abbreviated as SLDNF-resolution.

[2] In fact the relationship between ASP and Turing completeness if far more subtle. A complete characterization of this relation fall outside of the scope of this paper. Roughly speaking, a "traditional" ASP implementations, where a programme is provided as a set of constraints covers the complexity class NP, but not the entire class REC of computable functions [28, 70]. For a complete characterisation of this relationship, we refer to [12].

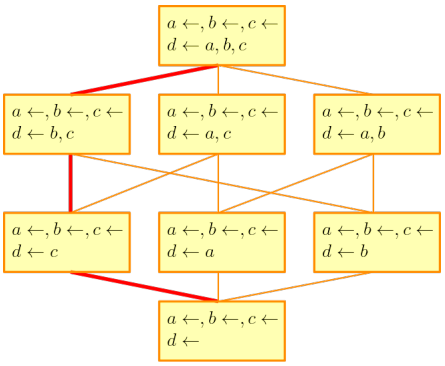


Figure 3.1: The hypothesis space of the 2^P-RESOLUTION abstract reasoning framework, where $P = \{a \leftarrow, b \leftarrow, c \leftarrow, d \leftarrow a, b, c\}$. With the red edges we denote the traversal followed by Prolog's SLD-resolution.

3.5 Conclusions

In this chapter we briefly discussed clausal logic. Clausal logic has been at the core of computational logic for middle of the last century. It also played a seminal role in the development of automatic theorem provers [33]. Another useful application of clausal logic is in logic programming languages such as Prolog, or Clingo, which may be regarded as restricted theorem provers[3]. Logic programming languages are widely used as a reasoning framework for artificial intelligence. In this paper we are interested in this last perspective. We saw that Clausal Logic can be utilised as a framework for computational deductive reasoning.

In Section 3.1 we presented the semantics for clausal logic. In Section 3.2 we introduced our main inference rule known as *resolution*. Then, in Section 3.3 we examined one of the most popular application of clausal logic, namely logic programming. We also considered a variant of resolution, that is SLD-resolution. Lastly in Section 3.4 we reviewed the hypothesis space induced by applying resolution on sets of Horn clauses.

[3] After all programmes are proofs, due to Curry-Howard isomorphism [74].

Chapter 4

Single Clause Induction

In this chapter we discuss induction on a single Horn-clause reasoning framework. Our search space will be finite and constituted from clauses of the form $\mathtt{h} \leftarrow \mathtt{b_1}, \ldots, \mathtt{b_n}$ for a *fixed* head atom $\mathtt{h}$, and a set of n possible body atoms $\mathtt{b_i}, i \in [n]$. In other words, our goal is to find a *single* definition of a notion described by $\mathtt{h}$. Our generality relation will be an approximation for the syntactic entailment $\vdash$ called *subsumption* $\vdash_{\text{sub}}$. Consider a finite set of possible body atoms $\mathcal{B}$ and a particular atom h. We call the resulting reasoning framework $(\mathcal{B} \cup h)$-Subsumption. This reasoning framework is suitable for computation, since subsumption does not only provide a theoretical relation between premise and conclusion, but also provides a *syntactic* way to reconstruct the premises form conclusion. We will be introducing subsumption gradually in the next two sections, by firstly presenting the propositional case and then we discuss subsumption for first-order clauses. Then we examine some of the lattice properties of our reasoning framework. In this chapter, we mainly follow the presentation of [61].

Before we proceed to the formal definitions, we will give an example on propositional subsumption. Note that, *intuitively* a clause of the form $\mathtt{h} \leftarrow \mathtt{b_1}$ is *more general* than the clause $\mathtt{h} \leftarrow \mathtt{b_1}, \mathtt{b_2}$. That is because the first clause demands *fewer* conditions for $\mathtt{h}$ than the second clause. In the following example we consider a simple example regarding the definition of a *human* according to a anecdotal depute between the Greek philosopher Plato and Diogenes.

Example 3 (On the Definition of a Human: Plato vs Diogenes)**.** *Consider that the atom for which we would like to find the definition is* $h \equiv \mathtt{human}$*. Additionally, consider a set of characteristics* $\mathcal{B} = \{\mathtt{featherless}, \mathtt{biped}, \mathtt{flat\_nails}\}$*. The search space of a definition for "human" is depicted in Figure 4.1. In an anecdote reported by Diogenes Laërtius, Plato defined the "human" as a "featherless biped". Then the cynic philosopher Diogenes of Sinope plucked a chicken and brought it to Plato's Academy saying "Behold! I've brought you a human", then the academy added "with broad flat nails" to the definition.*

Note that the search space considered in Example 3 has both *minimum* and *maximum* elements, namely $\mathtt{human} \leftarrow$ and $\mathtt{human} \leftarrow \mathtt{featherless}, \mathtt{biped}, \mathtt{broad\_nails}$ respectively. The former being the most general clause (denotes that *everything* is a human), while the latter being the most specific. Additionally, we observe that the search space is a *lattice*. This is no coincidence, we will see in the sequel that the search space of $(\mathcal{B} \cup h)$-Subsumption is always a lattice[1].

Example 3 also provides a useful intuition for the *limitations* of Single Clause Induction. The examples in such a system, when the head of our clause is fixed, would be simple atoms. In more detail, when $b^+ \in E^+$, then b would be a characteristic of a human. Thus b^+ can be considered as a candidate for the body of our unknown clause. On the other hand when $b^- \in E^-$, then b^- is a characteristic not shared by any human. Therefore we are prohibited to use it in the definition. In Example 3 Diogenes "cheats" since he does not brings a negative example to Plato that is used in his definition, but an *unwanted consequence*. Consider the first definition by Plato,

$$\pi_1 \equiv \mathtt{human} \leftarrow \mathtt{featherless}, \mathtt{biped}. \tag{4.1}$$

Also, consider the *common knowledge* clauses utilised by Diogenes,

$$\begin{aligned} \delta_1 &\equiv \mathtt{featherless} \leftarrow \mathtt{plucked\_chicken}, \\ \delta_2 &\equiv \mathtt{piped} \leftarrow \mathtt{plucked\_chicken}. \end{aligned} \tag{4.2}$$

[1] Possibly *after* adding the special elements $\top, \bot$, as we discussed in Section 1.2.

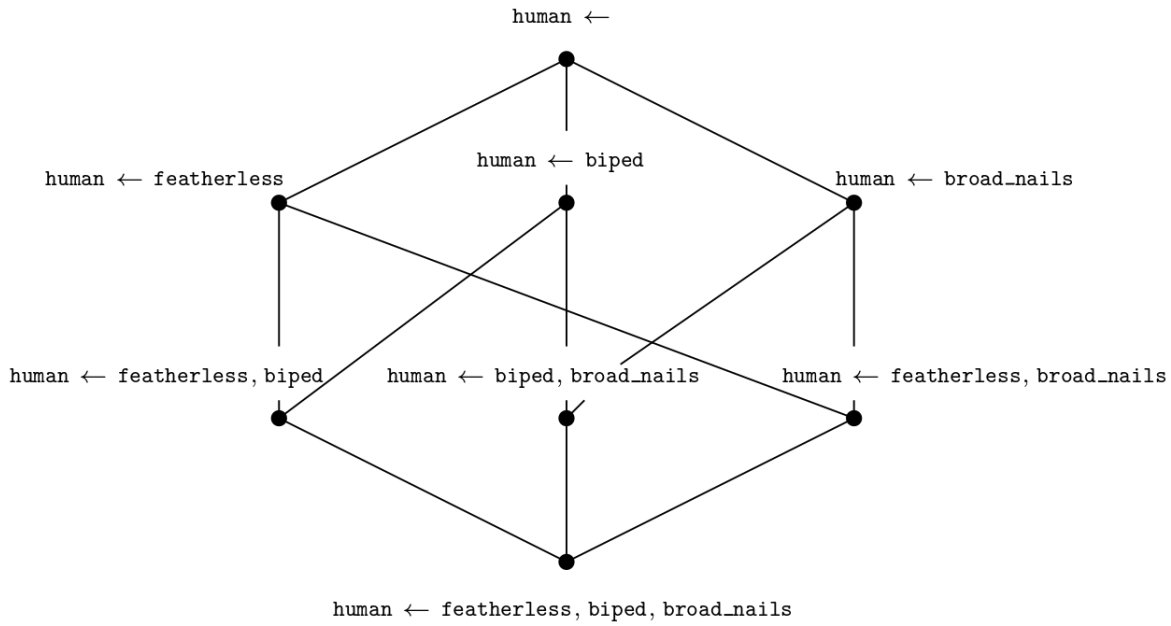


Figure 4.1: The search space for Plato's definition for the "human".

Of course, using subsumption (or any of the entailment operators discussed so far) we cannot conclude δ_1 or δ_2 from π_1, i.e. $\pi_1 \not\models_{\text{sub}} \delta_1$ or $\pi_1 \not\models_{\text{sub}} \delta_2$. What Diogenes implicitly suggests is that when π_1 is used together with the common knowledge facts δ_1, δ_2 leads to the absurdity, that is,

$$\{\pi_1, \delta_1, \delta_2\} \models \texttt{human} \leftarrow \texttt{plucked\_chicken}. \tag{4.3}$$

We observe in (4.3) that we need to take into account *sets* of clauses to argue about Diogenes witty response to Plato's first definition. We leave this discussion for Chapter 5.

4.1 Propositional Subsumption

In this and the following sections we gradually define the semantics for the *subsumption* and the reasoning framework $(\mathcal{B} \cup h)$-SUBSUMPTION. Of course, as in all reasoning frameworks discussed thus far we have a propositional and a first-order variant. We first consider the mechanics of subsumption introduced in the propositional setting. Then we discuss the subtleties when dealing with variables and first order atoms. There we make use of the *most general unifier* defined in Chapter 3. Later, the two characteristics are combined in order to define subsumption for first order Horn clauses.

We begin with an example in propositional logic. Assume the Horn formula $\phi_1 \equiv a \leftarrow b, c, d$. Naturally, we have $\phi_1 \models\!\models a \vee \neg b \vee \neg c \vee \neg d$. Now, let ϕ_2 be an arbitrary formula, where we omitted some of the literals, we will always have $\phi_2 \models \phi_1$, or $\phi_2 \succeq \phi_1$. For example, $\phi_2 \equiv a \leftarrow b, c$. In this case we say that ϕ_2 *subsumes* ϕ_1. Now, let us use the set notation introduced in Chapter 3 for clauses. We would have $c_{\phi_1} = \{a, \neg b, \neg c, \neg d\}$, while $c_{\phi_2} = \{a, \neg b, \neg c\}$. Observe that $c_{\phi_2} \subseteq c_{\phi_1}$. Following this intuition we define subsumption formally bellow.

Definition 11 (Propositional Subsumption)**.** *Let $\phi_1, \phi_2 \in$ HORN$_{PL}$ be two propositional Horn clauses. Additionally, consider their corresponding set representations c_1, c_2. We say that ϕ_1* subsumes *ϕ_2, notation $\phi_1 \vdash_{sub} \phi_2$, if and only if $c_1 \subseteq c_2$, i.e.*

$$\phi_1 \vdash_{sub} \phi_2 \Leftrightarrow c_1 \subseteq c_2, \tag{4.4}$$

Note that, from computational point of view, (4.4) introduces *non-determinism*. Consider an inductive setting where we want to compute a generalization for a clause ϕ_2 using the subsumption operator $\vdash_{\text{sub}}$. Subsumption suggest to erase some of the atoms at the body of ϕ_2, in order to construct ϕ_1. When ϕ_1 has n atoms to its body, this gives us 2^n options for generalisation. This, naturally, results in a computational bottleneck.

4.2 Subsumption in First Order Atoms

We now proceed to the realm of first order logic. When dealing with clauses in first order logic, we need to also take into consideration variables and thus unification. Consider the simplest first order language, that only includes first order atoms of the same "name", i.e. instances of the same predicate. We call this framework $(\varnothing \cup h)$-Subsumption, since we regard clauses with the same predicate as head and no body.

Definition 12 (Subsumption in First Order Atoms)**.** *Let a_2, a_1 be two clauses of $(\varnothing \cup h)$-Subsumption. We say that a_1* subsumes *a_2, notation $a_1 \vdash_{sub} a_2$, when there is a substitution θ, such that $a_1\theta \equiv a_2$.*

Note that in the above definition subsumption coincides with simple *unification*. For example the atom $\texttt{mother}(\texttt{X}, \texttt{Y})$ subsumes the atom $\texttt{mother}(\texttt{padme}, \texttt{leia})$[2]. On the other hand, $\texttt{father}(\texttt{X}, \texttt{luke})$ does not subsume $\texttt{father}(\texttt{vader}, \texttt{leia})$.

When dealing with inductive reasoning instead of substituting variables with terms, we follow the opposite direction. Namely, $\texttt{mother}(\texttt{padme}, \texttt{leia})$ is generalised to $\texttt{mother}(\texttt{X}, \texttt{Y})$, since $\texttt{mother}(\texttt{X}, \texttt{Y}) \models \texttt{mother}(\texttt{padme}, \texttt{leia})$, and thus $\texttt{mother}(\texttt{X}, \texttt{Y}) \succeq \texttt{mother}(\texttt{padme}, \texttt{leia})$. Such a substitution is called *inverse substitution*. Let a be an atom and θ a substitution. The inverse substitution θ^{-1} satisfies,

$$a\theta\theta^{-1} \equiv a. \tag{4.5}$$

When utilising inverse substitutions we need to be careful when a term is used multiple times inside an atom. For example, a *minimal* inverse substitution of the atom $\texttt{p}(\texttt{a}, \texttt{b}, \texttt{f}(\texttt{a}))$ should be $\texttt{p}(\texttt{X}, \texttt{Y}, \texttt{f}(\texttt{X}))$, instead of e.g. $\texttt{p}(\texttt{X}, \texttt{Y}, \texttt{Z})$. While both latter atoms are generalizations of the former, it should be intuitive that $\texttt{p}(\texttt{X}, \texttt{Y}, \texttt{f}(\texttt{X}))$ constitutes a better choice, with respect to minimality. We discuss further this observation in the next subsection, regarding the *least general generalization* operator.

We close this discussion of subsumption on first order atoms with two important remarks. Firstly note the presence of infinite chains in the hypotheses space, if function symbols are allowed. Indeed, consider the sequence $\texttt{p}(\texttt{X}), \texttt{p}(\texttt{f}(\texttt{X})), \texttt{p}(\texttt{f}(\texttt{f}(\texttt{X}))), \ldots$. Each previous element in this infinite sequence subsumes the next. These infinite chains innate the danger of invoking infinite loops in an inductive reasoning algorithm. On the other hand, the use of variables introduces syntactic variants. Observe that $\texttt{p}(\texttt{X}) \models\!\!\models \texttt{p}(\texttt{Y})$. We say that an atom a_2 is a *variable renaming*[3] of an atom a_1, when there are substitutions θ, θ', such that $a_1\theta \equiv a_2$ and $a_2\theta' \equiv a_1$, and both θ, θ' are 1-1 variable substitutions. Fortunately, it can be shown that two atoms can be syntactic variants only if they are variable renaming of each other [61]. These syntactic variants may also lead an inductive system to infinite loops.

4.2.1 Lattice properties of $(\varnothing \cup h)$-Subsumption

Using subsumption we may introduce a lattice structure on the search space of first order atoms. In order to do that, and as mentioned above, we need to consider equivalence classes of variable renaming. In order to argue that this structure is indeed a complete lattice, we need to define the greatest lower bound (glb) and least upper bound (lub) operators for any two atoms. For glb, assume a_1, a_2 be two atoms, we need to find some atom a that is subsumed by both a_1, a_2. Let θ be the *most general unifier* of a_1, a_2 then,

$$a_1 \vee a_2 = a_1\theta \equiv a_2\theta. \tag{4.6}$$

It is easy to see that, indeed when θ is the most general unifier of a_1, a_2 and $a \equiv a_1\theta \equiv a_2\theta$, then $a_1 \vdash_{\text{sub}} a$ and $a_2 \vdash_{\text{sub}} a$, while a is the *greatest* element with that property, with respect to $\vdash_{\text{sub}}$. Observe that from (4.6) glb corresponds to unification. Symmetrically, lub will correspond to *anti-unification*, using the inverse substitutions introduced in equation (4.5). A simple example of the structure induced in $(\varnothing \cup h)$-Subsumption is given bellow.

Example 4 (Lattice of $(\varnothing \cup h)$-Subsumption)**.** *The lattice induced for a simple $(\varnothing \cup h)$-Subsumption structured is presented in the Figure 4.2. Here we assume $h \equiv \texttt{pred}/2$. We disallow the use of*

[2] Recall that in clausal logic we assume implicitly universal quantification, thus $\texttt{mother}(\texttt{X}, \texttt{Y})$ is an abbreviation for $\forall X, \forall Y$ mother(X, Y). Therefore, subsumption in first order clausal logic is sound.

[3] The reader familiar with λ-calculus may notice that variable renaming corresponds to α-reduction. λ-calculus also suffers from syntactic variants and α-reduction, or α-renaming, is used to introduce equivalence classes.

functional symbols. Note that when allowing a single functional symbol the lattice becomes infinite. Additionally, we assume the Herbrand universe to consists of only two constants, namely $U = \{\mathtt{a}, \mathtt{b}\}$. *In* $(\varnothing \cup h)$-SUBSUMPTION *we will always have a maximum element, that is* h. *Observe that in order to induce a complete lattice, we need to include the special element* $\bot$. *Otherwise we have multiple minimal elements. We used the technique introduced in Chapter 1, in equations (1.5) and (1.6).*

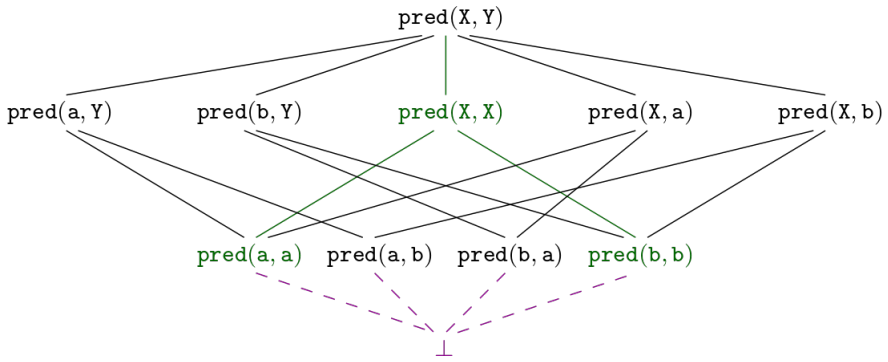


Figure 4.2: The lattice induced in $(\varnothing \cup h)$-SUBSUMPTION, for $h \equiv \mathtt{pred}/2$.

Least General Generalization

The glb operator corresponds to the *least general generalisation (lgg)* and is of particular importance in inductive logic programming. We briefly discuss this operator here.

First we consider terms. Recall that a term is compound by function symbols variables and constants. Assume t_1, t_2 be two terms, for $\text{lgg}(t_1, t_2)$ we have,

$$\text{lgg}(t, t) \equiv t, \tag{4.7}$$
$$\text{lgg}(f(s_1, \ldots, s_n), f(t_1, \ldots, t_n)) \equiv f(\text{lgg}(s_1, t_1), \ldots, \text{lgg}(s_n, t_n)), \tag{4.8}$$
$$\text{lgg}(f(s_1, \ldots, s_m), g(t_1, \ldots, t_n)) \equiv V, \tag{4.9}$$
$$\text{lgg}(s, X) \equiv V. \tag{4.10}$$
$$\text{lgg}(X, t) \equiv V. \tag{4.11}$$

Equations (4.7)-(4.11) need some explaining. In (4.7) we assume that $t_1 \equiv t_2$. In (4.8), each term constitutes of the same function symbol with the same arrity. In (4.9) we assume that $f \not\equiv g$, while V is a "fresh" variable not appeared in either of the terms. Lastly, (4.10), (4.11) are symmetrical, in both cases X is a variable, and V is a fresh variable with $V \not\equiv X$. When a fresh variable is used it becomes *associated* with that particular lgg of the two terms t_1, t_2. Thus, when in the computation $\text{lgg}(t_1, t_2)$ reappears it should be substituted with the *same* variable. For example $\text{lgg}(\mathtt{f(a,a)}, \mathtt{f(b,b)}) \equiv \mathtt{f(X,X)}$, since we associate the variable X with $\text{lgg}(a, b)$ from the first argument, and reuse it in the second.

We use the equations (4.7)-(4.11) recursively in order to compute the lgg of atoms.

$$\text{lgg}(p(s_1, \ldots, s_m), p(t_1, \ldots, t_n)) \equiv p(\text{lgg}(s_1, t_1), \ldots, \text{lgg}(s_n, t_n)) \tag{4.12}$$

4.3 θ-subsumption

In this section we combine the notion of subsumption in propositional logic, and first order atoms, in order to introduce a more general relation in first order Horn clauses, namely θ-subsumption. θ-subsumption is the most important framework for generalization in inductive logic programming. Almost all inductive logic programming systems use it in one form or another.

Definition 13 (θ-subsumption)**.** *Assume two first order clauses* $\phi_1, \phi_2 \in \text{HORN}_{FOL}$. *Also, consider their corresponding set notation representations* c_1, c_2. *We say that* ϕ_1 θ-subsumes ϕ_2, *notation* $\phi_1 \vdash_{sub} \phi_2$, *when there is a substitution* θ *such that* $c_1\theta \equiv c_2$.

Note that θ-subsumption is *sound* that is, if ϕ_1 θ-subsumes ϕ_2, then $\phi_1 \models \phi_2$. On the other hand, θ-subsumption is *not* complete. Fortunately it is only incomplete for *self-recursive* clauses. Consider, for example, $\phi_1 \equiv \texttt{nat(s(X))} \leftarrow \texttt{nat(X)}$, which models the simple fact that the successor of a natural number is a natural number, and $\phi_2 \equiv \texttt{nat(s(s(Y)))} \leftarrow \texttt{nat(Y)}$, which models that even numbers are natural. We can easily check that $\phi_1 \models \phi_2$. But from Definition 13 $\phi_1 \not\vdash_{\text{sub}} \phi_2$.

Since θ-subsumption combines the properties of both propositional subsumption and subsumption in first order atoms, it unfortunately inherits both their complexity bottleneck, namely the choice of subset and substitution θ. Therefore, testing θ-subsumption is NP-complete [27]. An immediate consequence is that checking for the coverage of a first order clause, namely which of the examples are covered, is also NP-complete. Since a θ-subsumption test is a computationally expensive operation, an efficient inductive system should minimize the number of the tests.

4.3.1 θ-subsumption Variants

We now briefly discuss some variants of θ-subsumption that aim to solve some of the problems mentioned above. Namely, *object identity*, or OI-subsumption and *reverse implication*. The first variant attempts to overcome the issue regarding syntactic variants, while the latter the incompleteness of θ-subsumption.

OI-subsumption. One of the difficulties with θ-subsumption is that a longer clause c_{ϕ_1} can θ-subsume a shorter c_{ϕ_2}. This may occur when there exist substitutions θ for which several literals of c_{ϕ_2} collapse into one. Object identity prevents this because it requires that all terms in a clause be different. In this framework a clause ϕ is completed to the clause com(ϕ), where is demanded that all distinct terms are different. In other words, we force syntactic difference to coincide with semantic difference. For example the clause,

$$\texttt{p(X,X)} \leftarrow \texttt{q(X,X)}, \texttt{r(Y,a)}.$$

Is completed to,

$$\texttt{p(X,X)} \leftarrow \texttt{q(X,X)}, \texttt{r(Y,a)}, \texttt{X} \neq \texttt{Y}, \texttt{X} \neq \texttt{a}, \texttt{Y} \neq \texttt{a}.$$

Given two clauses ϕ_1, ϕ_2, ϕ_1 OI-subsumes ϕ_2, if and only if there exist a substitution θ such that, com(c_{ϕ_1})$\theta \subseteq$ com(c_{ϕ_2}). OI-subsumption is (strictly) *weaker* than θ-subsumption. It is, therefore, a weaker approximation of logical entailment. Additionally, two clauses are equivalent under OI-subsumption if and only if they are variable renamings. This eliminates the need for equivalence classes. Finally, the least general generalization of two clauses may not be unique. We rewrite the example of Figure 4.2, when we assume $\texttt{X} \neq \texttt{Y}$ in Figure 4.3.

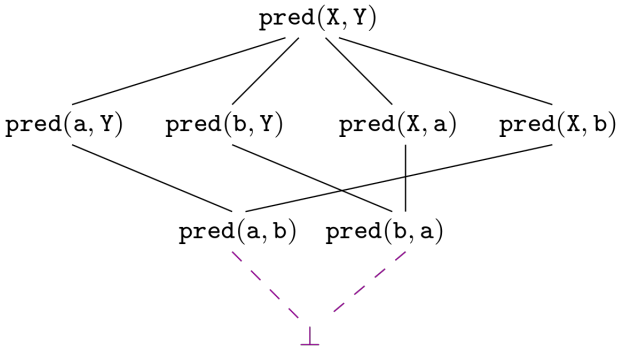


Figure 4.3: The lattice of Figure 4.2, when implicitly assume $\texttt{X} \neq \texttt{Y}$.

Reverse implication. Reverse implication addresses the incompleteness of θ-subsumption with respect to logical entailment. Rather than inverting θ-subsumption this framework attempts to invert the implication relation among clauses. The implication framework differs from θ-subsumption only for recursive clauses. The computational properties of the framework are worse than those of θ-subsumption because testing whether one Horn clause logically implies another is only *semi-decidable* [43]. The least general generalization, for Horn clauses, under reverse implication, is either not unique, or not a Horn clause. Because of these difficulties the framework of inverse

implication is seldom used in practice. Furthermore, given the completeness of θ-subsumption for non-recursive clauses, it is only relevant for applications requiring recursion, such as program synthesis.

4.3.2 Lattice Structure of $(\mathcal{B} \cup h)$-Subsumption

We now briefly discuss the lattice structure that appears in the $(\mathcal{B} \cup h)$-Subsumption framework for first order Horn clauses, with the same head. Note that the greatest lower bound and least upper bound are direct extensions from the operators discussed in the previous sections of this chapter. As we will see in the next example it is often required to include the special elements top $\top$ or bottom $\bot$. Note the expansion of the search space by considering substitutions. The search space under θ-subsumption becomes *exponentially* bigger from its propositional counterpart. Lastly, observe that

Example 5 (Lattice of $(\mathcal{B} \cup h)$-Subsumption)**.** *In Figure 4.4 we present the* semi *lattice structure of* $(\mathcal{B} \cup h)$-Subsumption*, with* $h \equiv \mathtt{p}/1$ *and* $\mathcal{B} = \{\mathtt{q}/1\}$*. It is important to note that we disallow the use of functional symbols. In our case the Herbrand universe is* finite*,* $U = \{\mathtt{a}, \mathtt{b}\}$*. Note that, when a single functional symbol is allowed, then the Herbrand universe is* infinite*. As we shall see in the sequel, we often disallow the usage of functional symbols. Also, observe the expansion of the search space by considering substitutions. The search space under* θ*-subsumption becomes* exponentially *bigger from its propositional counterpart. Additionally, note the usage of the bottom element in order to force the lattice property. Here, we used the technique introduced in Chapter 1, in equations (1.5) and (1.6). Lastly, we remark that in* $(\mathcal{B} \cup h)$-Subsumption *we will always have a maximum element namely* $h \leftarrow$*. On the other hand, we will usually not have a single minimum element.*

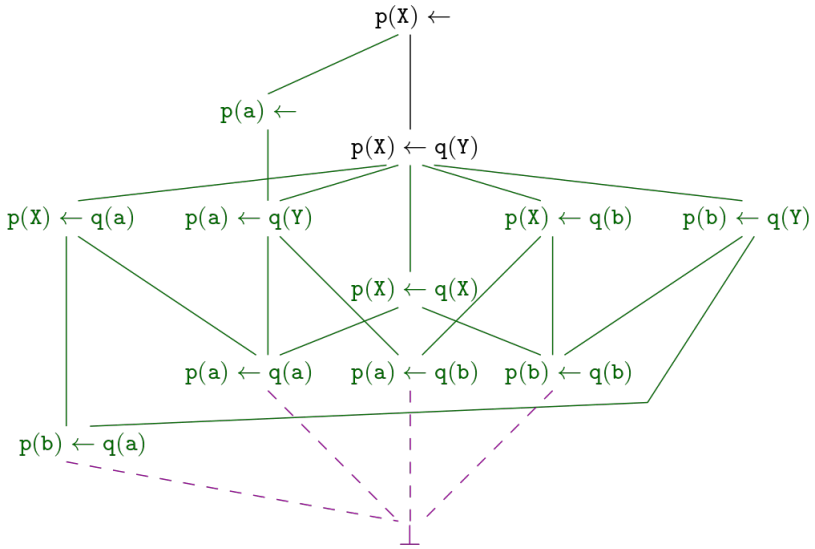


Figure 4.4: The lattice induced in $(\mathcal{B} \cup h)$-Subsumption, for $h \equiv \mathtt{p}/1$ and $\mathcal{B} = \{\mathtt{q}/1\}$. With green we denote the clauses obtained using substitution, while with black we denote the clauses obtained by adding conditions. With purple we denote the bottom element $\bot$, and the corresponding edges.

4.4 Conclusions

In this chapter we discussed induction in single clause hypothesis space. We introduced the $(\mathcal{B} \cup h)$-Subsumption reasoning framework. In this framework the generalization relation is given by subsumption $\vdash_{\text{sub}}$. We gradually defined the properties of the operator, firstly for propositional languages, namely when $\mathcal{B}, h \subset \mathsf{PL}$. There we observed the structural properties of $\vdash_{\text{sub}}$, that is generalization by omitting conditions and specialisation by adding conditions. Subsequently, we examined the reasoning framework $(\varnothing \cup h)$-Subsumption that includes first order atoms, of the same predicate. There we examined the properties of subsumption, when taking variables under consideration. We saw that subsumption corresponds to unification. We also, remarked the lattice

properties of the hypothesis space, possible by adding the special element $\bot$. Lastly, we considered subsumption in its entirety, for first order Horn clauses that use the same predicate as head.

Since the hypothesis space considered are lattices, possibly after expanding them with $\top, \bot$, the join ($\wedge$) and meet ($\vee$) operators are well defined. The join ($\wedge$) operator for the least upper bound is of special importance in the field of inductive logic programming. We called this operator *least general generalization*.

It is important to note the limitation of the approach we followed here. Single clause induction is best suited for discovering concept definitions, as we noted in the beginning of this chapter. Nevertheless, as we noted before, since the examples must follow the same format as the Horn clause to be discovered, some of the unwanted properties of our definition cannot be eliminated. We discuss induction for set of clauses in the following chapter.

Chapter 5

Multiple Clause Induction

In this chapter we introduce induction for multiple clauses, or set of clauses. A synonym term in literature for multiple clause induction is *theory revision*[1], since we consider sets of formulas, i.e., theories, which we want to revise in order to fit our examples. We begin by reconsidering Example 3 from Chapter 4. The first definition of a human being is given in (4.1), while the counter examples of Diogenes are given in (4.2). In Figure 5.1 we present the reasoning that leads to the absurdity of (4.3).

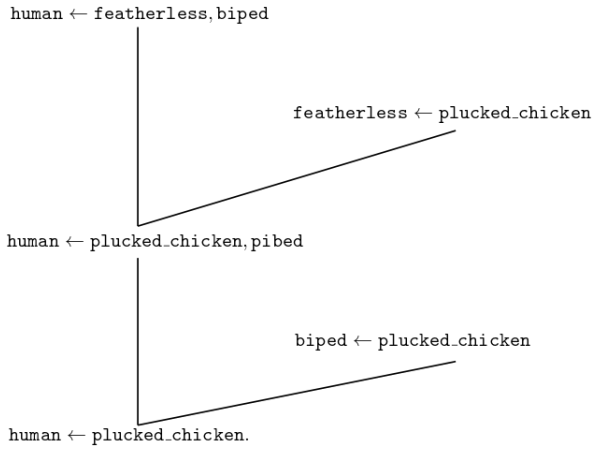


Figure 5.1: The resolution steps in the definite programme $\{\pi_1, \delta_1, \delta_2\}$. By applying consecutive resolution steps, we arrive at the absurdity of (4.3).

Observe that we applied consecutive *resolution* steps in order to arrive to the absurdity, from the *set* of clauses $\{\pi_1, \delta_1, \delta_2\}$, of equations (4.1), (4.2). This example provides two core elements for reasoning in a multi-clause setting. As we discussed in Chapter 3, the resolution operator $\vdash_{\text{res}}$ may induce an abstract reasoning framework on a space constituted by sets of clauses. The hypothesis space we traversed is depicted in Figure 5.2. The reasoning framework depicted in Figure 5.2 is a special case of 2^{HORN}-RESOLUTION introduced at the end of Chapter 3. Note that, since we handle sets of clauses, we essentially are using the operators (1.9) (1.10) for *theory revision* introduced in Chapter 1. Thus, for two theories[2] T_1, T_2 if T_1 is more general than T_2, i.e., $T_1 \succeq T_2$, if $T_1 \supseteq T_2$ or for every $t_2 \in T_2 \setminus T_1$ there is some $t_{1,1}, t_{1,2}$ such that $t_{1,1}, t_{1,2} \vdash_{\text{res}} t_2$[3].

In the sequel of this chapter we will discuss induction in a multi-clause setting. We base our presentation mainly on [61]. Our discussion will be essentially dived into three parts. Firstly, in Section 5.1 we discuss generalisation operators that *inverse* resolution. There we will see two important families of operators, namely `V` and `W` operators. Then, in Section 5.2, we will discuss how we may utilise *background knowledge* in an inductive system. Observe that we may rewrite the Horn programme of Figure 5.2 as $\pi_1 \cup \{\delta_1, \delta_2\}$, where $\{\delta_1, \delta_2\}$ constitutes a set of background knowledge already provided to the inductive system. Such knowledge may guide the search of the

[1] Not to be confused with *belief revision*.
[2] Here we use the terms "theory", "programme" interchangeably to refer to a set of Horn clauses.
[3] Note that $\vdash_{\text{res}}$ is a *binary* specialisation operator.

hypothesis space. Lastly, in Section 5.3, we move away from resolution and examine *abductive* logic. This will provide as with an alternate route for logic-based induction.

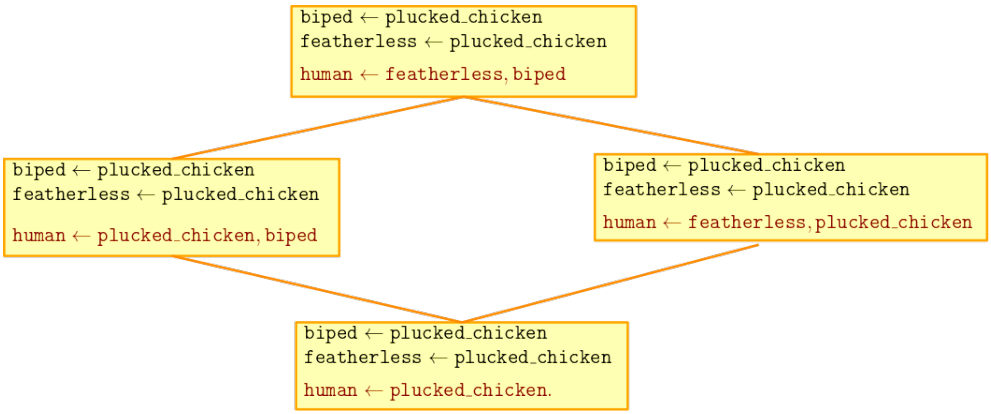


Figure 5.2: Part of the hypothesis space of the $2^{\{\pi_1,\delta_1,\delta_2\}}$-RESOLUTION reasoning framework. Here we only used the binary specialization operator induced by $\vdash_{\text{res}}$. We omitted theories of the hypothesis space constructed by discarding or adding clauses.

5.1 Inverse Resolution

In the previous chapter we presented various operators for generalization. We followed each time the same methodology, we introduced a relation to syntactically estimate the logical entailment $\models$, we argue that this relation organises the search space as a partial order, or a lattice, then for generalization we just invert this relation. Here we examine John Alan Robinson's *resolution*, introduced in Chapter 3. We discuss two operators for generalization based on resolution, namely `V`-*operator* and `W`-*operator*. The work on these operators were primarily influenced by the observations of Stephen Muggleton. Again, we follow the presentation of [61].

5.1.1 `V`-operator

There are two distinct `V`-operators, that invert Robinson's resolution, namely *absorption* and *identification*. Recall that resolution involves three clauses ϕ_1, ϕ_2, ϕ, from (3.3) we have $\phi_1 \wedge \phi_2 \vdash \phi$. In this case the general clauses are ϕ_1, ϕ_2, while ϕ is the specialization. Note that without loss of generality we can rewrite ϕ_2 as a conclusion, that is $\phi_1 \wedge \phi_2 \vdash \phi \wedge \phi_2$. The inverse resolution `V`-operators now induce ϕ_1, ϕ_2 from ϕ, ϕ_2.

The *absorption* operator proceeds according to the following scheme.

$$\frac{\begin{array}{l} h \leftarrow c_1, \ldots, c_{i-1}, g, c_i, \ldots, c_m \\ g \leftarrow b_1, \ldots, b_n \end{array}}{\begin{array}{c} h \leftarrow c_1, \ldots, c_{i-1}, b_1, \ldots, b_n, c_i, \ldots, c_m \\ g \leftarrow b_1, \ldots, b_n \end{array}} \tag{5.1}$$

A graphical representation of the `V`-operator is depicted in Figure 5.3. In this scheme the first clause bellow the line is the resolvent of the two clauses above the line and the second clause bellow the line is just a copy of the second clause above the line. Thus the specialization operator deductively infers the clauses below the line from those above the line, whereas the absorption operator inductively infers the clauses above the line from those below it.

Similarly, by changing the position of the copied clause, we obtain the *identification* operator.

$$\frac{\begin{array}{l} h \leftarrow c_1, \ldots, c_{i-1}, g, c_i, \ldots, c_m \\ g \leftarrow b_1, \ldots, b_n \end{array}}{\begin{array}{c} h \leftarrow c_1, \ldots, c_{i-1}, b_1, \ldots, b_n, c_i, \ldots, c_m \\ h \leftarrow c_1, \ldots, c_{i-1}, g, c_i, \ldots, c_m \end{array}} \tag{5.2}$$

So far, we discussed absorption (5.1) and identification (5.2) operators in propositional logic. As always, first order logic must also take into account variables and thus unification. Assume

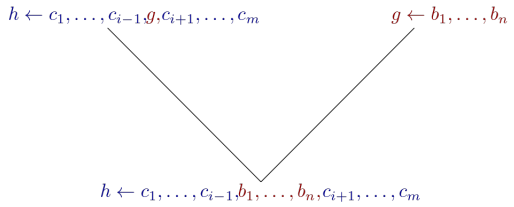
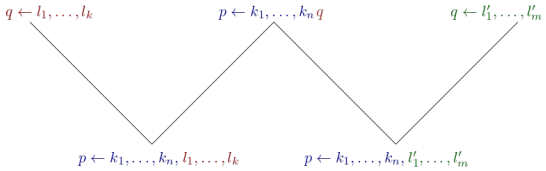
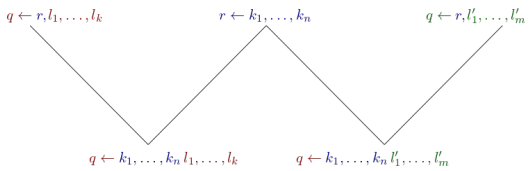


Figure 5.3: The V-operator for inverse resolution.



(a) The intra-construction operator of equation (5.3).



(b) The inter-construction operator of equation (5.4).

Figure 5.4: W-operators for inverse resolution. The above operators are used by inductive systems for *predicate invention*. In both figures the middle clause is induced by the system.

the first order clauses in set notation $c_1 = \{h, \neg g, \neg l_1, \ldots, \neg l_n\}$ and $c_2 = \{g', \neg l'_1, \ldots, l'_m\}$. The resolvent is the clause $c = \{h, \neg l_1, \ldots, \neg l_n, \neg l'_1, \ldots, l'_m\}\theta$, where θ is the most general unifier of g, g', i.e. $g\theta \equiv g'$. If we want to compute c_1 from c, c_2 we have to invert θ as well. Note that in general the inverse substitution θ^{-1} need not be unique. The different possible choices to invert substitution introduce further choice points and non-determinism in the search, which complicates the search strategy further. In [51] there is a detailed analysis of substitution inversion.

5.1.2 W-operator & Predicate Invention

There are also the so-called W-operators[4], which invert two resolution steps in parallel. The W-operator links two V-operators in such a way that one of the clauses is shared. Using the notation for inference operators, we obtain the following schemes for *intra-construction* (see (5.3)) and *inter-construction*, respectively (see (5.4)).

$$\frac{\begin{array}{l} q \leftarrow l_1, \ldots, l_k \\ p \leftarrow q, k_1, \ldots, k_m \\ q \leftarrow l'_1, \ldots, l'_n \end{array}}{\begin{array}{l} p \leftarrow l_1, \ldots, l_k, k_1, \ldots, k_m \\ p \leftarrow l'_1, \ldots, l'_n, k_1, \ldots, k_m \end{array}} \quad (5.3) \qquad \frac{\begin{array}{l} p \leftarrow r, l_1, \ldots, l_k \\ r \leftarrow k_1, \ldots, k_m \\ q \leftarrow r, l'_1, \ldots, l'_n \end{array}}{\begin{array}{l} p \leftarrow l_1, \ldots, l_k, k_1, \ldots, k_m \\ q \leftarrow l'_1, \ldots, l'_n, k_1, \ldots, k_m \end{array}} \quad (5.4)$$

The W-operators are of special interest because they *introduce new predicates*. For intra-construction operator (see (5.3)) the predicate q does not appear below the line, and hence when using this scheme inductively a new predicate will be introduced. For the inter-construction operator (see (5.4))the predicate r is new. The automatic introduction of new terms or predicates in the description language is called *predicate invention*. Inventing *relevant* new predicates is one of the hardest tasks in inductive logic programming, because there are so many possible ways to introduce such predicates. In Figure 5.4a we present graphically the intra-construction operator, while in Figure 5.4b, we depict the inter-construction operator.

The above-sketched complications in the search process explain why inverse resolution operators are not very popular in practice. Nevertheless, inverse resolution is extremely useful as a general framework for inductive reasoning. It can serve as a general framework in which other frameworks can be reformulated.

[4]In this case terminology is the most appropriate since the letter W, as its name suggests is the unification of two "V"s. Note that in Latin there where no distinction between the sounds of "u" and "v". Hence, the W-operators combine two V-operators.

5.2 Background Knowledge

As we discussed in the beginning of this chapter multi-clause induction allows us to consider the implications of *background knowledge* in an inductive system. Recall the problem statement discussed in Section 1.3. There we presented a variation of the inductive problem, where a system is provided with some existing knowledge B. Then, the goal of the inductive system is to expand this knowledge to some hypothesis $H \supseteq B$ that explains the examples. In a computational logic setting the background knowledge B is a knowledge base consisted of formulas. Providing background knowledge to a system often improves the systems accuracy, as we saw in Example 3. Researchers since Plotkin [57] have highlighted the advantages of including background knowledge in the inductive process and has become the norm in inductive systems.

Various frameworks have been developed, e.g. relative subsumption [58], relative implication [54] and generalized subsumption [10]. These differ mainly in the form of inference rule and derivation that is allowed starting from g and B. In the sequel of this section we briefly mention *bottom clauses*, and *semantic refinement*, while focusing in *relative least general generalisation*.

Bottom clause. The bottom clause is the most specific clause, within the hypothesis language $\mathcal{H}$, that covers an example with regard to the background theory. Formally, let B be a background theory, and ϕ a clause, the bottom clause $\bot(\phi)$ is the most specific clause, such that,

$$B \wedge \bot(\phi) \vdash \phi \tag{5.5}$$

The bottom clause is of interest since it bounds the search space from below. Any other hypothesis ψ, such that $\bot(\phi) \succeq \psi$ will *not* cover ϕ, and thus can be pruned away. The bottom clause captures all information that is relevant to the example and the background theory. Therefore, an inductive system may follow a top to bottom direction, starting from the top element $\top$, using a refinement operator that considers only generalization of the bottom clause $\bot(\phi)$. Such a strategy is deployed by Muggleton's Progol [50].

Semantic refinement. Most inductive logic programming algorithms proceed from general to specific while applying a refinement operator. When working under θ-subsumption care has to be taken that refinements of a clause are *proper*, i.e., refinements are not logically equivalent to the original clause. This problem becomes even more apparent when refining clauses under a background theory. Refinements ϕ' of ϕ, such that $B \wedge \phi' \models \phi$ are redundant, and therefore should not be considered during the search process. When applying a refinement operator under θ-subsumption, this amounts to requiring that ϕ' does not θ-subsumes $\bot(\phi)$. Thus, after computing a refinement clause, we can check the refinement for redundancy, using the bottom clause. Note that in order to optimise this task the bottom clause need not be computed explicitly.

5.2.1 Relative Least General Generalization

We now discuss the existence and computation of least general generalization (lgg) with respect to a background theory. Recall that lgg (see equations (4.7)-(4.11)) is one of the most inportant notions when dealing with generalization on a lattice structured hypothesis space. We denote the relative lgg with respect to a background theory B $\text{rlgg}_B(\cdot,\cdot)$. In order to compute $\text{rlgg}_B(\phi_1,\phi_2)$ of two clauses ϕ_1,ϕ_2 we can simply compute the lgg of the corresponding bottom clauses, i.e.,

$$\text{rlgg}_B(\phi_1,\phi_2) = \text{lgg}(\bot(\phi_1),\bot(\phi_2)). \tag{5.6}$$

The special case where the language of background knowledge $\mathcal{B}$, and the language of examples $\mathcal{E}$ consists *only* of ground atoms is considered by the inductive logic programming system Golem [53]. Assume an instance of a background knowledge base $B \in \mathcal{B}$, we also denote, without confusion with B the conjunction of all ground facts in the background theory. Also, assume two (positive) examples $e_1, e_2 \in E$, in our set of examples $E \in \mathcal{E}$. Then, the computation of $\text{rlgg}_B(\cdot,\cdot)$ is simplified to,

$$\text{rlgg}_B(e_1,e_2) = \text{lgg}(e_1 \leftarrow B, e_2 \leftarrow B). \tag{5.7}$$

5.3 Abduction

In this section we move away from resolution and introduce a new reasoning operator, namely *abduction* denoted with $\vdash_{\text{ab}}$. Abduction can be regarded as a special case of consequtive resolutions. Namely, consider the Horn clause $c \leftarrow a, b$ and the facts $a \leftarrow, b \leftarrow$. Then, we have $(c \leftarrow a, b), (a \leftarrow) \vdash_{\text{res}} c \leftarrow b$ and $(c \leftarrow b), (b \leftarrow) \vdash_{\text{res}} c \leftarrow$. Thus, combining the facts $a \leftarrow, b \leftarrow$, with the Horn rule $c \leftarrow a, b$, we derive the new fact $c \leftarrow$. In classical logic, we can rewrite the above derivation as $(c \leftarrow a, b), (a \wedge b) \vdash c$. The latter formulation is exactly the abduction operator for propositional logic. In the equation (5.8), bellow, we present the abduction operator for first-order formulas.

$$\frac{\begin{array}{l} q_1\theta \wedge \cdots \wedge q_n\theta \\ p \leftarrow q_1, \ldots, q_n \end{array}}{\begin{array}{l} p\theta \\ p \leftarrow q_1, \ldots, q_n \end{array}} \tag{5.8}$$

Note that in equation (5.8) θ is considered a given substitution for the facts $q_1, \ldots, q_n$. Of particular interest is the case, when θ is a *ground substitution* for $q_1, \ldots, q_n$, namely when the variables have been eliminated from the facts. Then, *if each variable of* p *appears in its body*[5], the resulting fact $p\theta$ is also a ground atom. Naturally, in our context we are mostly interested in the inversion of (5.8). Assume that an inductive system has already derived the clause $p \leftarrow q_1, \ldots, q_n$ and $p\theta$ is an example. Then we can infer the additional examples $q_1\theta, \ldots, q_n\theta$. Since we will consider Horn clauses, we will refer to the resulting reasoning framework as 2^{Horn}-Abduction. Observe that, since abduction is a special case of resolution it holds that 2^{Horn}-Abduction $\subseteq 2^{\text{Horn}}$-Resolution

Abduction can be used to address *dependencies* in theory revision. Recall that in our setting a theory is essentially a set of clauses. Using equation (5.8) we can reduce examples for one predicate to those for other predicates. By revising the theory to correctly handle these other examples, the original theory revision problem can be solved. We examine the following cases:

- If $p\theta$ is a *positive example* and the clause $p \leftarrow q_1, \ldots, q_n$ belong to the theory, then we infer that the facts $q_1\theta, \ldots, q_n\theta$ are positive examples.
- If $p\theta$ is a *negative example* and the clause $p \leftarrow q_1, \ldots, q_n$ belong to the theory, then we infer that *at least one* of the facts $q_1\theta, \ldots, q_n\theta$ must also be negative.

The first case is a direct application of the abductive inference operator. The second case corresponds to its *contra-position*. It is especially useful when the clause is used in a proof of $p\theta$ from the theory. Indeed, the proof will no longer hold if one of the $q_i\theta$ no longer follows from the theory. We illustrate the use of abduction in the following example.

Example 6 (Logic Programming Debugger)**.** *Using abduction we may define an elegant debugger based on logic programming. Assume a (possibly declarative programme) P consisting of the subroutines $\mathtt{r}_1, \mathtt{r}_2, \mathtt{r}_3$ and the main routine $\mathtt{m}$. Now assume that the predicates $\mathtt{m}/2, \mathtt{r}_1/2, \mathtt{r}_2/2, \mathtt{r}_3/2$ correspond to the input-output relation of the respective routines. For example, the predicate $\mathtt{r}_1(\mathtt{X}, \mathtt{Y})$ holds when the routine $\mathtt{r}_1$ takes as input $\mathtt{X}$ and outputs $\mathtt{Y}$. Then, the input-output relation of the whole programme P can be written as a Horn formula,*

$$\mathtt{m}(\mathtt{X}, \mathtt{Y}) \leftarrow \mathtt{r}_1(\mathtt{X}, \mathtt{Z}), \mathtt{r}_2(\mathtt{Z}, \mathtt{W}), \mathtt{r}_3(\mathtt{W}, \mathtt{Y}). \tag{5.9}$$

Now assume that there is an instance denoted by the substitution θ such that $\mathtt{m}\theta$ is wrong, i.e. given the instance does not compute the desired output. Then, the debugger, using abduction, will perform test on each $\mathtt{r}_\mathtt{i}\theta$, knowing that some of the routines is faulty. Alternatively, when $\mathtt{m}\theta$ is correct, we know that each $\mathtt{r}_\mathtt{i}\theta$ is correct[6]*.*

Of particular interest in our setting is when, in the above example the programme P is a *definite programme*, written in a logic programming language such as Prolog. There P is a set of Horn clauses. The notion of a programme and the set of clauses (theory) that describes its behaviour coincide. When $P \vdash_{\text{ab}} \mathtt{m}\theta$ is a wrong derivation, then we know that some of the definitions of $\mathtt{r}_\mathtt{i}/2$ is wrong. We also know that the fault happens for the input substitution θ. In this sense, abduction can be used to create new examples. In a *query based* setting these examples will correspond to queries to an oracle or the user. Such a system is the Model Inference System developed by Shapiro in 1991 [66], which we discuss in Chapter 6.

[5] A Horn clause where each of the heads variables appears also in its body, is called *rage restricted*. In practice, in computational logic, when we discuss Horn clauses, we implicitly assume that they are rage restricted.

[6] Naturally, this do not inform us about the global correctness of the programme, for each instance.

5.4 Conclusions

In this chapter we discussed induction for multiple clauses. In this setting we are interested in finding a set of clauses, or theory that explains a given set of examples. In Section 5.1 we reconsidered the 2^{Horn}-Resolution reasoning framework for induction. There we examined two kinds of operators, namely the `V`-operators and the `W`-operators. The latter category of inductive operator is of special interest to our setting, since allows for *predicate invention*. The 2^{Horn}-Resolution framework, also, allows as to provide an inductive system with existing background knowledge. Thus, our goal is now to extend the existing knowledge to a theory $H \supseteq B$ that covers the given examples. We discussed background knowledge for inductive systems in Section 5.2. Lastly, in Section 5.3 we presented a refinement of resolution for facts, namely *abduction*. Abductions enables us to derive new examples from existing ones that our induced theory must cover. These additional examples can be used to test the correctness of an existing theory.

Chapter 6

Implementing Induction

In this chapter we review various implementations of the inference systems presented in Chapters 4, 5. Note that thus far we only implicitly discussed the algorithms behind the methods we studied. As we shall see in Section 6.1 we can derive a simple generic algorithm for traversing the hypothesis space. This algorithm will take as parameters a generalisation operator $\rho(\cdot)$, a hypothesis language $\mathcal{H}$ and possibly a heuristic ranking function $z(\cdot)$. In the previous chapters we focused on the hypothesis languages $\mathcal{H}$ and the generalisation operators $\rho(\cdot)$. Changing these parameters induces a different inductive system. In Section 6.2 we will consider additional restrictions to the hypothesis language $\mathcal{H}$, that aim to ease some of the computational burden. In the sequel we focus our attention to three interesting implementations of inductive systems. In Section 6.3 we review the First Order Inductive Learner (FOIL). In Section 6.4 we examine a query-based, *polynomial* propositional system. In Section 6.5 we present the seminal work by Shapiro, the Model Inference System. Lastly, in Section 6.6 we conclude this chapter and briefly discuss some other implementations, including the most recent advances in the field of Inductive Logic Programming.

6.1 Generic Inductive Algorithm

Here we present some abstract strategies for the GeneralInduction-SingleHypothesis (IS) and GeneralInduction-MultiHypotheses (IM) problems introduced in Section 1.3 at the beginning of this paper. Essentially, we consider these problems as one. As we will see in the sequel of this chapter IM can either be considered as a series of IS sub-problems, or as an IS on *sets*. Our goal in this introductory section, is to provide an intuition on the methods employed by the systems we will later review. Thus, we will leave the details on the variants of IS, or IM for the later sections. For the same reason, we will not consider background knowledge. We will examine two symmetrical algorithms, namely the Generic-TopDown algorithm and the Generic-BottomUp algorithm. As their name suggest they follow a top-down and a bottom-up trajectory of the search space respectively. In order to gain a deeper understanding of their mechanics we provide an example.

Example 7 (Single Target Concept: Defining a Bird)**.** *Assume we would like to come up with a dictionary definition of the word* "bird"*. We take as an example a parrot. We know that a parrot can fly, has feathers and can mimic sounds. Since a parrot is a bird, we construct the set of positive examples using the above attributes, namely* $E^+ = \{\texttt{fly}, \texttt{feathers}, \texttt{mimic}\}$*. On the other hand, we know that not every bird is able to fly, e.g. a penguin or a chicken. Additionally, very few birds are able to mimic the voices they hear. Thus, we compose the set of negative examples as following* $E^- = \{\texttt{fly}, \texttt{mimic}\}$*.*

Since, we already have an instance of a bird in mind, we already know a very specific *definition of a bird, namely the parrot. This definition has as head the* target concept *and as body each characteristic of* E^+*. Namely,*

$$e^+ \equiv \texttt{bird} \leftarrow \texttt{fly}, \texttt{feathers}, \texttt{mimic}.$$

On the other hand, we know that not all birds can fly, thus the definition $\texttt{bird} \leftarrow \texttt{fly}$ *is wrong. Additionally, the definition* $\texttt{bird} \leftarrow \texttt{mimic}$ *is also wrong, since only some of the sub-species of parrots have that ability. Now, suppose* h *is our resulting hypothesis we demand* $h \models \texttt{bird} \leftarrow \texttt{fly}$

and $h \not\models \texttt{bird} \leftarrow \texttt{mimic}$*, which is logically equivalent to* $\neg(h \models \texttt{bird} \leftarrow \texttt{fly}, \texttt{mimic})$*. Thus, we construct the following clause, as our counter-example,*

$$e^- \equiv \texttt{bird} \leftarrow \texttt{fly}, \texttt{mimic}.$$

Therefore, we are looking for a hypothesis h *such that* $h \succeq e^+$*, but* $h \not\succeq e^-$*. We depict the search space of the induction problem discussed here in Figure 6.1.*

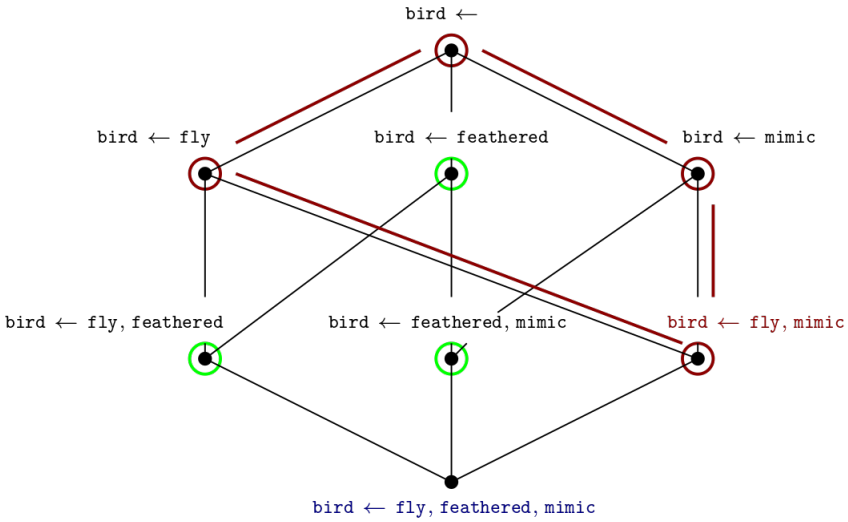


Figure 6.1: The search space of Example 7. With red we denote the restricted area of the negative example e^-. With blue we denote the positive example e^+. Lastly, with green we denote the *feasible area*.

Since we re-interpreted the definition of the IS problem and highlighted the correspondence for our logic-based setting, we are now ready to present an algorithm that solves it. We begin by presenting two useful subroutines, namely Specialization and Generalization bellow.

Specialize(H, $\rho_s(\cdot)$, $z(\cdot)$, E^+)	
Input:	H, the current hypothesis. $\rho_s(\cdot)$, a specialization operator. $z(\cdot)$, a heuristic ranking function. E^+, the set of positive examples.
Output:	A specialization H' of H
return s.t.	$\arg\max z(H')$ for each $e^+ \in E^+$, $H' \succeq e^+$, $H' \in \rho_s(H)$.

Generalize(H, $\rho_g(\cdot)$, $w(\cdot)$, E^-)	
Input:	H, the current hypothesis. $\rho_g(\cdot)$, a generalization operator. $w(\cdot)$, a heuristic ranking function. E^-, the set of negative examples.
Output:	A generalization H' of H
return s.t.	$\arg\max w(H')$ for each $e^- \in E^+$, $H' \not\succeq e^-$, $H' \in \rho_g(H)$.

Observe the symmetry in the two above routines. Essentially, in each of the above routines we solve an *optimisation problem on posets* [36]. Each routine takes as argument the current hypothesis H. The Specialize routine takes as argument a specialization operator $\rho_s(\cdot)$, while the Generalize takes the corresponding generalization operator $\rho_g(\cdot)$. Note that these operators will *syntactically* construct a set of potential candidates each. They correspond to the *smallest* step of specialization and generalization respectively. Additionally, in each method we consider some heuristic functions, namely $z(\cdot), w(\cdot)$ that rank the candidate hypotheses. Note that in some systems the computation of the ranking takes into account the set of examples, e.g. $z(H) = z(H, E^+, E^-)$, this is not reflected to our presentation in order to ease the notation. Lastly, in each routine we traverse the hypothesis space with respect to a set of *constraints*. Namely, when we specialise the hypothesis we must be careful to not miss any of the positive examples. On the other hand, when we generalize, we must be cautious not to explain any negative examples.

Below, we present a top-down traversal algorithm. Note that we simplified our presentation, by abstracting away the parameters of the Specialize subroutine. In Algorithm 1 we begin from the top element and we perform consecutive specializations. The main while-loop stops when the

Algorithm 1: Generic-TopDown

```
H ← ⊤
while |H| < |E⁺| − ε do
    H ← Specialize(H)
return H
```

length of our hypothesis becomes relatively range, with respect to the set of positive examples (recall our discussion on Kolmogorov complexity in Chapter 1).

Next we present a generic bottom-up traversal method in Algorithm 2. The Generic BottomUp algorithm is symmetrical to Algorithm 1, except from the fact that we start from the *most specific hypothesis* $\bot(E^+)$ (see Section 5.2), which is not included in the training examples. Thus, some prepossessing is needed. On the other hand, the computation of the most specific hypothesis follows a similar generalization-based process [11, 50]. In Example 7 the bottom clause is essentially e^+. In Algorithm 2 observe the symmetry of the stop criterion. Intuitively, since we start from the most specific clause, our initial hypothesis will be "too descriptive". Thus we continue our search until a certain level of compactness, with respect to the size of training examples.

Algorithm 2: Generic-BottomUp

```
H ← ⊥(E⁺)
while |H| ≥ |E⁺| − ε do
    H ← Generalize(H)
return H
```

Recall the search space of the Example 7, depicted in Figure 6.1. When we employ a top-down search following Algorithm 1, we will firstly arrive at the clause `human` ← `feathered`. The clause we will return depends on the choice of the parameter ε. For example, for $\varepsilon = 1$ we return `human` ← `feathered`. On the other hand, for $\varepsilon \geq 2$ it is possible to return some of the other clauses of the feasible area, with two atoms in their body. This might contradict our intuition, since we established that not all birds can fly, or mimic; how can a more specific definition be feasible? This has to do with the definition of the GenericInduction-SingleHypothesis problem, and the whole setting of induction in an abstract reasoning system. Since we demand a hypothesis h not to explain a negative example e^-, we essentially force a certain level of specialization on the hypothesis. In other words, the negative examples form an upper bound to the search space, with respect to the partial order $\succeq$. Contrarily, the positive examples form a lower bound of the feasible space. Moreover, when the hypothesis space is a *lattice*, the search space will consist of the *sub-lattice* induced from all the generalizations of $\bot(E^+)$. The parameter ε essentially defines the *distance* between the most specific clause $\bot(E^+)$ and a desired generalization. Namely, ε induces a even tighter lower bound to the search space. The impact of the parameter ε on the search space is depicted in Figure 6.2. Lastly, observe in Figure 6.2 the difference between the two kinds of restraints used in this inductive setting. The restraints induced by the two sets of examples respect the partial order, thus being more refined. On the contrarily, the constraint induced by the ε parameter follows an implicit metric space on the poset, dictated by the length (or *descriptive complexity*) of the terms.

6.1.1 Induction Complexity

We now make some brief remarks on the *computational complexity* of the algorithms examined above. Suppose that we apply Algorithm 1 on an an abstract reasoning framework $\mathcal{R} = \langle \mathcal{H}, \succeq \rangle$. Let $G = \texttt{height}(\mathcal{R})$ be the *height* of the poset. Also, let S be the *maximum number of specializations* of a hypothesis $H \in \mathcal{H}$. Note that $S \leq \texttt{width}(\mathcal{R}) = W$. We assume that the complexity of the ranking function $z(\cdot)$ is *polynomial* on the size of its input, i.e. $\texttt{comp}(z) = \text{poly}(S) = O(\text{poly}(W))$. Additionally, assume that the complexity of the specialization operator is polynomial on the size of its *output*, namely $\texttt{comp}(\rho_s) = \text{poly}(S) = O(\text{poly}(W))$. Moreover, with $\texttt{comp}(\succeq)$ we denote the complexity of computing the comparison between two hypotheses. Lastly, let $E = |E_1| + |E_2|$.

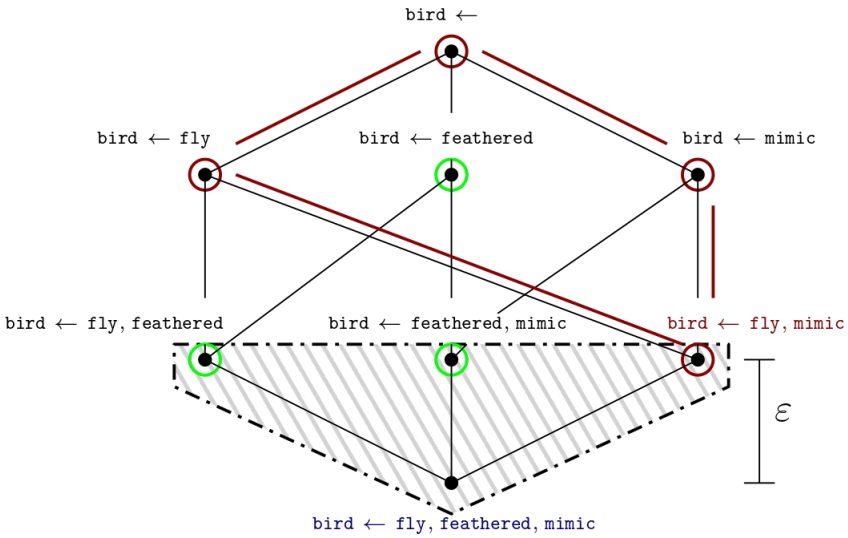


Figure 6.2: The search space of the Example 7, under the additional constraint induced by ε. The grey area marks the additional heuristic restriction on the feasible area by ε.

Note that $E \leq |\mathcal{H}| \leq G \cdot W$[1]. Then, the complexity of the SPECIALIZE subroutine is bounded above, from the following complexity.

$$O\left(\text{comp}(\rho_s(\cdot)) \cdot S \cdot \text{comp}(z(\cdot)) \cdot E \cdot \texttt{comp}(\succeq)\right) \tag{6.1}$$

By applying the assumptions and inequalities sketched previously we arrive at the following result.

Theorem 6 (Computational Complexity of Algorithm 1)**.** *Assume a abstract reasoning framework $\mathcal{R} = \langle \mathcal{H}, \succeq \rangle$. Additionally, let $\langle E^+, E^- \rangle$ be the sets of positive and negative examples respectively. Finally, let $\rho_s(\cdot)$ be a specialization operator and $z(\cdot)$ be a ranking function. Then, Algorithm 1 will find an optimal hypothesis $H^\star$ in at most,*

$$O\left(G \cdot \textit{poly}(W) \cdot \texttt{comp}(\succeq)\right) \tag{6.2}$$

Naturally, due to symmetricity, equation (6.2) also gives an upper bound to the worst time complexity of Algorithm 2. Observe that in the above discussion we assumed that our reasoning framework was *finite*. This does not hold for reasoning frameworks based on a subset of the first order logic, e.g. when the Herbrand universe is infinite. For this reason many inductive systems avoid an infinite Herbrand universe, by disallowing non-constant function symbols. Observe that, from eq. (6.2) the complexity is dictated by the form of the reasoning framework. On the other hand, the underlying poset depends on the choice of the hypothesis language. Now the difficulty of the induction problem should become apparent. Often, the parameters $W, \texttt{comp}(\succeq)$ are *exponential*. Consider the simplest, finite, first order language we examined in Chapter 4, i.e. $(\mathcal{B} \cup h)$-SUBSUMPTION. Then, from Sperner's lemma $W = O\binom{|\mathcal{B}|}{|\mathcal{B}|/2}$ [68], while the complexity of the subsumption relation $\vdash_{\text{sub}}$ is an NP-hard problem. Lastly, under subsumption there may appear infinite chains and thus $G \to \infty$. These negative results showcase the need of additional *heuristic* constraints to be forced on the underlying hypothesis language in order for the induction problem to ever be tractable. Further results on the complexity of induction and relative results on PAC-learnability of inductive learners can be found in [16].

6.1.2 An Algorithm for Query-Based Induction

We close this section by reviewing an alternative view on the induction problems. Contrary to our earlier discussion, where a fixed set of examples are provided to our system, here we assume that the algorithm has access to an oracle function $\texttt{orac}(\cdot)$. Recall, that in Section 1.5 we discussed a framework where the oracle function $\texttt{orac}(\cdot)$ evaluates the sign of an example, whether it is positive or negative. In general the information provided by the oracle function should be enough

[1]For any (finite) poset $\mathcal{P} = \langle P, \succeq \rangle$ it holds that $|P| \leq \texttt{height}(\mathcal{P}) \cdot \texttt{width}(\mathcal{P})$.

to substitute the information provided by the examples. Here we present a more sophisticated framework. We assume that an adversary keeps an unknown hypothesis $H^\star \in \mathcal{H}$. We are allowed to ask queries of the form $\texttt{comp}(H, H^\star)$, where H is our current hypothesis. The $\texttt{comp}(\cdot,\cdot)$ oracle will inform us about the relationship between our current hypothesis H and the unknown true hypothesis $H^\star$. Since the second argument of $\texttt{comp}(\cdot,\cdot)$ will be fixed throughout the execution of an instance, we may define the comparison oracle as $\texttt{comp}\colon \mathcal{H} \to \{\succeq, \prec, \not\sim\}$, returning whether our current hypothesis H is more general, more specific, or not compared to the true hypothesis $H^\star$. Note that since, $\mathcal{E} \subseteq \mathcal{H}$ this model is a *generalization* of the one introduced in Section 1.5.

Observe that here we may assume that the set of positive E^+ and negative E^- examples is initially *empty*. The system will generate the examples it needs. Given a hypothesis H we partition the hypothesis space $\mathcal{H}$ in the following sets,

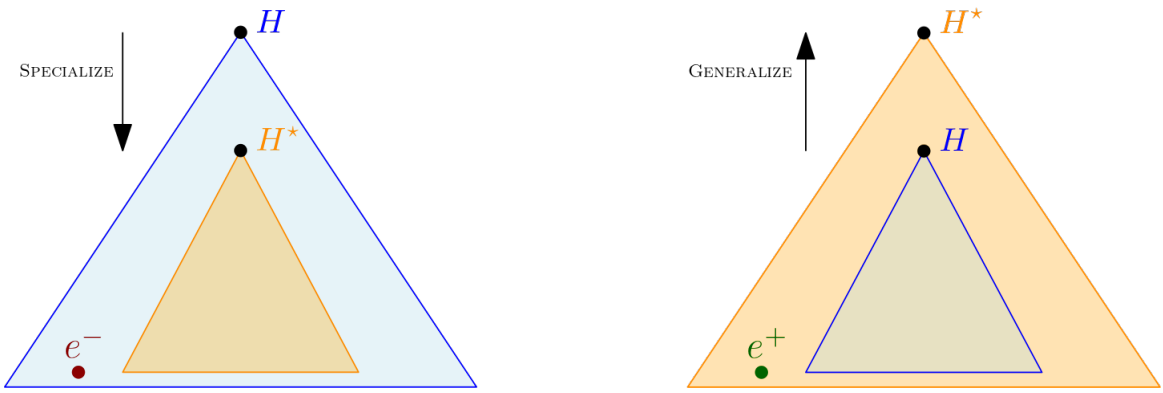
$$\mathcal{H}_{\succeq H} = \{H' \in \mathcal{H} \mid H' \succeq H\}, \tag{6.3}$$
$$\mathcal{H}_{\prec H} = \{H' \in \mathcal{H} \mid H' \prec H\}, \tag{6.4}$$
$$\mathcal{H}_{\not\sim H} = \{H' \in \mathcal{H} \mid H' \not\sim H\}. \tag{6.5}$$

The eq. (6.3) represent the *generalizations* of H (inclusive). The eq. (6.4) represent the specializations of H. Finally, eq. (6.5) represent the hypotheses that are unrelated to H. Naturally, since $\mathcal{E} \subseteq \mathcal{H}$, we can define similarly the sets $\mathcal{E}_{\succeq H}, \mathcal{E}_{\prec H}, \mathcal{E}_{\not\sim H}$, which also form a partition of $\mathcal{E}$. For the examples, it is useful to consider the sets $\mathcal{E}^+_H = \mathcal{E}_{\succeq H}$, $\mathcal{E}^-_H = \mathcal{E}_{\prec H} \cup \mathcal{E}_{\not\sim H}$ denoting the sets of positive and negative examples, respectively. In Algorithm 3 we present our generic algorithm for query-based induction.

Algorithm 3: Generic Query-Based Induction

```
H ← INITIALISE(H)
while |H| > 1 do
    if comp(H, H*) =≽ then
        H ← H \ H_{≻H}                          /* Prune away H_≻ */
        FIND e⁻ ∈ E⁺_H ∩ E⁻_{H*}
        E⁻ ← E⁻ ∪ e⁻
        H ← SPECIALIZE(H)
    else if comp(H, H*) =≺ then
        H ← H \ H_{≺H}                          /* Prune away H_≺ */
        FIND e⁺ ∈ E⁻_H ∩ E⁺_{H*}
        E⁺ ← E⁺ ∪ e⁺
        H ← GENERALIZE(H)
    else
        /* comp(H, H*) =≁ */
        H ← BACKTRACK(H)
return H
```

Observe that in this subsection we utilize the framework of [20] discussed in Chapter 1 for *posets*. Our algorithm implicitly assumes that the unknown hypothesis $H^\star$ exists in our hypothesis space $\mathcal{H}$ and *identifies* it, thus essentially performing a search on the poset induced by $\mathcal{H}$. We begin by choosing an arbitrary initial hypothesis H. Then we perform consecutive queries that aim to sequentially bound the search space by discovering new constraints induced by both the discovered examples and the queries to the oracle. If the $\texttt{comp}(H, H^\star)$ query returns that our current hypothesis is more general than the unknown hypothesis, then we need to specialize our hypothesis. Thus, all the generalizations of our current hypothesis can be eliminated from the search space, hence our set of hypotheses is reduced to $\mathcal{H} \setminus \mathcal{H}_{\succ H}$. Additionally, there must be an example e^- explained by H but not by $H^\star$ (see Figure 6.3a). We find such an example and add it to our negative examples. Lastly, we use the subroutine SPECIALISE to update our current hypothesis, with respect to the newly found example e^-. In the case when $\texttt{comp}(H, H^\star) = \prec$ we move symmetrically (see Figure 6.3b). Finally, there is the case when the current hypothesis H and $H^\star$ cannot be compared. In this case we need to *backtrack* and reconsider our last step. If in the previous iteration we performed a generalization, we need to choose an alternative generalization.



(a) When our hypothesis is too general, there is a negative example e^- that is explained, while it should not.

(b) When our hypothesis is too specific, there is a positive example e^+ that is not explained, while it should.

Figure 6.3: The Specialize and Generalize operations of Algorithm 3.

Symmetrically if our previous step was a specialization. If our search space is a lattice through backtracking we will be able to find the correct generalization (respectively specialization) and continue our search. Note that in the case $\texttt{comp}(H, H^\star) = \succeq$ we only prune away the proper generalizations of H, in order to be able to backtrack if needed.

Equivalence & Membership Queries

While we hope that Algorithm 3 helps the reader to develop intuition behind the search in a query-based setting, in practice there are some particularities that need to be addressed that were not included. Specifically the Find command in lines 6 and 11. For instance, in line 6 it is unrealistic to assume that the algorithm may find an example e^-, such that $e^- \in \mathcal{E}^+_H \cap \mathcal{E}^-_{H^\star}$, since $H^\star$ is *unknown*. This setback is not trivial, since it forces us to reconsider the choice of the oracle function. In literature there have been proposed two families of queries that strengthen our setting, namely *equivalence* and *membership* queries. A membership query is the "sign" query we discussed at the beginning of this section and in Chapter 1. A membership oracle, $\texttt{memb}\colon \mathcal{E} \to \{+, -\}$ will return the "sign" of a given example, with respect to the unknown hypothesis $H^\star$. On the other hand, an equivalence query is much more informative. An equivalence oracle $\texttt{equiv}\colon \mathcal{H} \to \{\text{"yes"}, \text{"no"}\} \times \mathcal{E}$[2] answers whether a hypothesis H is *logically equivalent* to the unknown hypothesis $H^\star$, that is $H \models\!\models H^\star$. If the equivalence holds, it will just return "yes". Contrarily, if the equivalence does not hold it will provide us with a *counterexample* $c \in (\mathcal{E}^+_H \setminus \mathcal{E}^-_{H^\star}) \cup (\mathcal{E}^-_H \setminus \mathcal{E}^+_{H^\star})$. If $H \succeq c$ then we know that c is a negative example, i.e. it is explained by our hypothesis, but it should not. Otherwise, we conclude that c is a positive example. In the first case we know that our hypothesis is too general, in the latter our hypothesis is too specific. Thus, we deduce the relation of our hypothesis with respect to $H^\star$. Additionally, the counter example is provided to us by the oracle. In a sense the Find command becomes incorporated to the oracle. Additional membership queries may be performed by the system in order to determine the correct generalization (respectively specialization) at each step.

A comprehensive survey on query-based formal concept learning can be found here [3]. It is interestingly to note that despite their intuitively similarity, membership and equivalence queries provide different irreplaceable functionality to an inductive system. Namely there are concept classes that cannot be learned efficiently using only membership or only equivalence queries, but can be learned when both types of queries are allowed. One such instance is induction on sentences of propositional Horn clauses[3]. Namely, there is no polynomial-time algorithm that exactly identifies any Horn sentence using only membership queries [1] or only equivalence queries [2]. Contrarily, as we see in Section 6.4 there is a polynomial algorithm that exactly identifies any Horn sentence, but uses both types of queries [4]. Despite the intuition we attempted to develop here, the exact relation between the membership, equivalence queries setting and comparison queries setting remains open.

[2] We assume that when the equivalence holds, the query just returns the *empty example* $\varnothing$.
[3] That is conjunction of propositional Horn clauses.

Conjecture 3 (Equivalence between membership, equivalence and comparison queries)**.** *There is a learning algorithm L that* exactly identifies *[4] an unknown hypothesis h of a known hypothesis space $\mathcal{H}$, using membership and equivalence queries, if and only if there is an algorithm L' that exactly identifies h using only comparison queries.*

6.2 Language Bias

In this section we will discuss various techniques that restrict the hypothesis language $\mathcal{H}$ and subsequently constraint the search space that needs to be explored by an inductive system. These methods are referred to by the literature as *language biases*. Syntactic biases restrict the form of the hypotheses, while semantic biases constrict the behaviour of a hypothesis. Here we will restrict our attention to syntactic biases. A syntactic bias can be stated elegantly as a formal grammar that specifies a certain subset $F \subseteq \mathcal{H}$ of the hypothesis language $\mathcal{H}$. A suitable choice of the subset language F will allow the inductive algorithm to focus on the desired hypotheses, while pruning away redundancies. We base our presentation in [61, 19].

6.2.1 Meta-rules

Consider the $(\mathcal{B} \cup h)$-Subsumption abstract reasoning framework of Chapter 4. As we noted in the previous section the complexity of an inductive algorithm depends on the number of candidate body atoms in $\mathcal{B}$, since the width of the underlying poset is bounded by $2^{|B|}$. Roughly this is because any subset of $\mathcal{B}$ may form the body of a hypothesis. Observe that if we constrict the number of atoms in the body of a clause to k, the upper bound for the width of the poset is now $\binom{k}{k/2}$. This constitutes a considerate reduction of the hypothesis space. Such constraints on the hypothesis language can be defined by using *meta-rules* or *clause-schemata*. Schemata are essentially second-order clauses, where the predicate names are replaced by predicate variables. For example, consider the following clause.

$$\texttt{P(X)} \leftarrow \texttt{Q(X,Y)}, \texttt{R(X)} \tag{6.6}$$

In the clause of eq. (6.6) the symbols $\texttt{P}, \texttt{Q}, \texttt{R}$ constitute *meta-variables* or second-order variables and can be substituted by predicates. For instance, the schema of eq. (6.6) can be instantiated to the clause $\texttt{mother(X)} \leftarrow \texttt{parent(X,Y)}, \texttt{female(X)}$.

6.2.2 Types

Another direction we can follow in order to restrict the search space is to consider *types*. The type of a predicate is declared by specifying the types of its arguments. For example we would write $\text{type}(\texttt{pred}(t_1, t_2, \ldots, t_n))$, to denote that in predicate $\texttt{pred}/n$ the i-th argument is of type t_i. In addition, there can be type definitions, which contain a specification of a *domain*, e.g. $\text{type} = [v_1, \ldots, v_m]$. Observe that by restricting the the types of each argument, we reduce the number of specializations (or generalizations) of a hypothesis.

6.2.3 Modes

A form of syntactic bias that combines elements of the above methods is *modes*. Mode declarations are considered the most popular variant of syntactic bias [19]. Mode declarations state which predicates may appear in a clause, how often, and also their argument types. In a mode language, `modeh` declarations denote which literals may appear in the *head* of a clause and `modeb` declarations denote which literals may appear in the *body* of a clause. Both type of declarations follow the same syntax, described bellow.

$$\texttt{mode}(\text{recall}, \texttt{pred}(\square t_1, \ldots, \square t_n))$$

In the notation above, recall denotes the number of times the predicate $\texttt{pred}/n$ can appear in the body of a clause. The recall parameter may take a positive integer value, or the wildcard $*$, meaning "any number". Additionally, the variables t_i denote the types of the corresponding arguments of the predicate $\texttt{pred}/n$. Lastly, the symbol $\square$ may take the values $\square \in \{+, -, \#\}$, which specifies if the corresponding argument is an *input*, *output* of *ground* argument respectively. For example, the declaration $\texttt{modeb}(2, \texttt{parent}(+\text{person}, -\text{person}))$ states the the predicate `parent`

takes two arguments of the type "person", the first of which is regarded as input, while the second is regarded as output. Finally, the latter definition allows for the predicate `parent` to appear only two times in the body of a clause.

6.3 FOIL: First Order Inductive Learner

In this section we review Quinlan's First-Order Inductive Learner or FOIL [60] from 1990. FOIL is one of the first symbolic inductive systems that was based on a subset of first order logic and thus falls in the category of ILP. It draws ideas from previous work on the propositional learners AQ [44], ID3 [59] and CN2 [15]. The FOIL system is a *relational* single concept learner, that learns first-order, function-free Horn clauses, and supports recursive definitions. Its goal is to provide a characterisation for a *single relation*, which may be consisting of multiple clauses (rules) that share the same predicate as the head. In order to compose a theory T, FOIL follows an incremental approach. At each step a new rule is learned and added to the theory. When a new rule is added to the theory, all the examples that it covers (i.e. explains) are removed from the data set. The examples are given in the form of *relations* or *tables*. The row of such a table correspond to instances of the corresponding relation[4]. FOIL uses a *closed word assumption*, thus considers any of the possible instances not included at the table as *negative* examples; hence all the instances included at the provided tables are considered positive.

Following the terminology introduced in this paper, FOIL solves a sequence of the General-Induction-SingleHypothesis (IS) problem instances, $I_0, I_1, \ldots, I_n$. After solving the instances I_k it will generate a new rule t_i which is added to the theory, i.e. $T_i = T_{i-1} \cup t_i$. Note that not all examples need to be covered in the i-th instance I_i, in order to proceed to rule generation. If all the examples are covered, then the algorithm stops and returns the generated theory T. In its i-th step, the rule generation procedure moves as in the traditional $(\mathcal{B} \cup h)$-Subsumption framework, employing a *top-down* approach, as in Algorithm 1. The set of candidate body predicates $\mathcal{B}$ consist of the relations provided to the algorithm. The head h consist of the *target relation*.

In order to generate a rule, FOIL employs a general-to-specific search strategy. It begins from the top element `target` $\leftarrow$, and continues by adding literals to the body of the rule. At each step of this inner loop, the best literal to be added is determined by the *weighted information gain* heuristic (wig). This heuristic plays the role of the ranking function $z(\cdot)$ we discussed in Section 6.1. Consider that the current rule is t, while t' is a specialization, i.e. $t \succeq t'$. Additionally, let E_t^+ denote the possitive examples covered by the rule t, symmetrically we define E_t^-. Then, the information gain heuristic is given by the following formula.

$$\text{wig}(t, t') = \frac{|E_{t'}^+|}{|E_t^+|} \cdot \left(\log \frac{|E_{t'}^+|}{|E_{t'}^+ \cup E_{t'}^-|} - \log \frac{|E_t^+|}{|E_t^+ \cup E_t^-|} \right) \tag{6.7}$$

Where with log we denote the logarithm with base 2. The term $\log \frac{|E_t^+|}{|E_t^+ \cup E_t^-|}$ is an estimate of the amount of information needed to specify that an example covered by the clause is positive. The difference between the two terms is the information gain. The information gain is then weighted by the fraction of positive examples that remain covered afgter the specialization. The weighted information gain balances information gain with coverage. The heuristic depicted in eq. (6.7) is a successor of a similar heuristic employed in the CN2 system.

6.4 The Horn Inductive System

In this section we discuss the Horn system by Dana Angluin et al. [4]. Horn is a *query-based* inductive system, that learns *propositional Horn sentences*. Thus, it constitutes a *multi-clause* system. In this setting an adversary is assumed, which holds an unknown Horn sentence $H^\star$. Our goal is to *exactly identify* $H^\star$ or a *logically equivalent* to $H^\star$ Horn sentence H', i.e. $H' \models\!\models H^\star$. The Horn system performs both *membership* and *equivalence* queries (see Section 6.1). A membership query returns the sign of an example. On the other hand, an equivalence querie answers whether our current hypothesis is equivalent to the unknown true hypothesis. If not, the equivalence oracle

[4]Here we use the terms *predicate* and *relation* interchangeably. Thus, we implicitly assume that the user essentially provides an *interpretation* to the inductive system in the form of tables (see Section 2.1). This implicit assumption is common in the area of relation databases.

returns a counterexample. Despite the limitations of its representation language, HORN provides useful insight to a query-based approach to induction.

The hypothesis space explored by HORN constitutes sets of propositional Horn clauses $2^{\text{HORN}_{\text{PL}}}$. This space is structured by the theory specialization operator induced by subsumption[5]. We will call the resulting reasoning framework $2^{\text{HORN}_{\text{PL}}}$-SUBSUMPTION. Due to its query based nature, HORN cannot be classified as a top-down or a bottom-up method. In the course of an execution the HORN algorithm will perform a series of generalizations and specializations depending on the answers received by the oracles. The algorithm begins from the top element, the *empty sentence*. In each step, makes an equivalence query on its current sentence H. If the response is "yes", then terminates and outputs its current hypothesis. Otherwise, the equivalence oracle will provide the algorithm with a counter example c. If $H \succeq c$ then c constitutes a negative example. HORN then finds the clause $h \in H$ that explains c and erase it from the current hypothesis H. Therefore making a specialization. If $H \not\succeq c$, then c is a positive example that is not covered by the current hypothesis. The algorithm then find the *most specific* clause that covers c and adds it to H. In order to do that HORN performs a series of membership queries. Thus making a generalization. This process terminates after a *polynomial* number of steps and queries.

Theorem 7 (HORN Complexity, Angluin et al. 1992 [4])**.** *For all positive integers m and n, the* HORN *algorithm* exactly identifies *every Horn sentence with m clause over n propositional variables in time $\widetilde{O}(m^3n^4)$[6] using $O(m^2n^2)$ equivalence queries and $O(m^2n)$ membership queries.*

An important corollary of Theorem 7 is that the class of Horn sentences is polynomial-time PAC-learnable if membership queries are available. We remark that HORN's setting is reasonable, since both types of queries, i.e. membership and equivalence, can be answered by the adversary in *polynomial time*, given the true hypothesis $H^\star$, *for propositional sentences*[7]. Additionally the authors in [4] present a second version of their algorithm that improves the complexities to $\widetilde{O}(m^2n^2)$ computational steps, $O(mn)$ equivalence queries, and $O(m^2n)$ membership queries.

6.4.1 Learning from Interpretations & Horn's Example Language

We conclude our discussions on the HORN system with some remarks on its example language $\mathcal{E}$. HORN follows a different approach from the one discussed in the previous chapters of this paper. An example in HORN is an *interpretation*, namely an assignment of truth values to the propositional variables. Equivalently, for a finite set of propositional variables V, then for an example e it holds that $e \subseteq V$. Recall from Chapter 2 that $e \models \phi$, where ϕ a logical formula when $e \in \mathcal{M}(\phi)$. Thus in a membership query, we exactly ask the adversary whether e belongs to $\mathcal{M}(H^\star)$. This setting for induction is called *learning from interpretations* [62], while the one mainly covered in this paper (and most popular in inductive logic programming [61]) is called *learning from entailment* [**plotkin**]. While the two models are equivalent, some systems choose one approach over the other. A variant of HORN that learns from entailment has been developed in [26].

6.5 The Model Inference System

In this section we briefly discuss the seminal work due to Shapiro, the MODEL INFERENCE SYSTEM [66], abbreviated henceforth to MODEL. MODEL infers first-order Horn theories from facts, i.e. grounded atoms. It is assumed that an adversary holds an unknown model $M^\star$. MODEL uses queries. It uses sign queries where an oracle answers whether a ground atom e is a positive or negative example, namely, whether $M^\star \models e$. Observe that given a model $M^\star$, we can partition the example space $\mathcal{E}$ to $\mathcal{E}^+, \mathcal{E}^-$, where for each $e^+ \in \mathcal{E}^+$ we have $M^\star \models e^+$ and $\mathcal{E}^- = \mathcal{E} \setminus \mathcal{E}^+$. Note that, both $\mathcal{E}^+, \mathcal{E}^-$ may be infinite. A theory $H \in \mathcal{H}$ is called a *complete axiomatization* of a model M if and only if $M \models H$ and $H \models \mathcal{E}^+$.

It is proved that MODEL can identify any first-order theory *to the limit* [30]. In this setting, a learner is presented with a sequence of facts $e_1, e_2, \ldots$. After observing a number of examples the

[5] See Section 1.4 for the relation between multihypothesis-single hypothesis specialization operators.

[6] In the $\widetilde{O}(\cdot)$ notation we omit the logarithmic terms.

[7] Note that for first-order Horn sentences an equivalence query becomes *undecidable*. Observe that, since a Prolog programme is essentially a Horn sentence and Prolog is Turing complete, then proving the logical equivalence of two first-order Horn sentences, is equivalent to proving the equivalence of two Turing machines. Form Rice's theorem we now the latter problem is undecidable. Additionally, even for general propositional sentences, an equivalence query is intractable, since it constitutes an co-NP-complete problem [40, 67].

learner, then, presents a *conjecture* (or hypothesis) that explains an initial part of this sequence. When an observation is presented that does not conform to the current hypothesis, the learner changes its conjecture. We say that the learner identifies the unknown theory to the limit, when after a point in time the learner does not change its conjecture. In other words, for an infinite sequence of examples, the learner needs to observe only a finite initial part of the sequence in order to explain it. Since MODEL identifies any theory to the limit, it returns a complete axiomatization of $\mathcal{E}^+$ with respect to the unknown model $M^\star$.

Note that in order for a complete axiomatization to be possible, i.e. the model inference problem to be solvable, the example language should be rich enough in order to refute any wrong hypothesis. Shapiro defines the notion of admissibility for a pair of languages $\langle \mathcal{E}, \mathcal{H} \rangle$. By the admissibility requirement, every false theory in $\mathcal{H}$ the implies $\mathcal{E}^+$ has a *witness* (namely a counterexample) in $\mathcal{E}$. This property couples the example and hypothesis languages together and implies that the difference in their expressive power should be *bounded* in such a way that every theory which is *successful*, i.e. implies all true examples and no false one, should also be *true*. The pair of ground atoms and Horn sentences constitutes such a pair and thus a complete axiomatization is possible.

MODEL operates roughly as HORN (see Section 6.4) for first-order Horn theories, under θ-subsumption. An each time it keeps a current hypothesis H. When a *negative* example is presented that does not conform to H, MODEL identifies the faulty clause $h \in H$ and specializes the theory by reducing it to $H \setminus h$. If the resulting theory is too specialized, MODEL generalizes it by adding to the theory a specialization h' of the previously refuted clause h. In order to identify the faulty clause, MODEL employs a sort of *abductive reasoning* [61]. When an uncovered positive example is presented, MODEL adds to the theory the most general clause that covers it, avoiding the previously refuted clauses.

Note that the specialization and generalization operators employed by MODEL are *complete* meaning that they are able to eventually explore the whole hypothesis space $\mathcal{H}$ in an ordered manner. The search space explored by MODEL is infinite since functional symbols are allowed. In practice this is infeasible and will result to large number of oracle-queries. For instance, in order to infer simple rules for addition and multiplication MODEL needs to be provided with 36 facts [66]. Thus the work of Shapiro on MODEL [66] is mainly of theoretical interest, providing an extensive theoretical framework for inductive reasoning. Additionally, it constitutes an important proof of concept and a turning point in the area of inductive reasoning. For example, the identification of the exact clause that is refuted by performing a series of experiments, was considered infeasible by the contemporarily philosophers of science methodology [66]. Therefore it represents a milestone in computational logic.

6.6 Conclusions

In this chapter we reviewed algorithms and implementations for inductive reasoning. We began in Section 6.1 by presenting some generic top-down and bottom up algorithms. Additionally, we introduced the concept of inductive learning through queries to an oracle. In the latter setting we presented another generic algorithm, but also discussed two popular types of queries, namely *membership* and *equivalence* queries. In Section 6.2 we presented various techniques for restricting the hypothesis space *syntactically*. These techniques are known in the literature as *language-biases* and facilitate in making the resulting inductive problem tractable. In each of the three subsequent sections we examined an algorithm and implementation for inductive reasoning. We start our case study, in Section 6.3, for the *first-order inductive learner*, or FOIL for short. FOIL constitutes a *greedy*, *incremental* algorithm that infers first-order, function free, Horn-clauses. FOIL follows a bottom-up strategy at each step generalizing a set of examples, creating a new rule (Horn clause) and add it to its current theory. As a ranking function for choosing between generalizations FOIL uses the *weighted information gain* heuristic. In Section 6.4 we consider the HORN system that infers propositional Horn sentences. HORN constitutes a query-based, *learning from interpretations* system that utilises both membership and equivalence queries. The HORN algorithm runs in polynomial time and query complexity, thus proving that the class of propositional Horn classes is PAC-learnable when membership queries are provided. Lastly, in Section 6.5 we survey the Model Inference System, abbreviated as MODEL. This algorithm infers a first-order Horn theory using sign queries (a variant of membership queries) to the user. It is proven that it can identify any first-order Horn theory *to the limit*. Despite its impressive theoretical results, MODEL's expressive power comes with a considerable computational cost. On the other hand, the work in MODEL

inspired many of the subsequent research in the area, providing a theoretical framework for the inductive reasoning.

6.6.1 Other inductive Systems

The goal of this chapter was to present some representative systems and implementations that covered a wide area of the Inductive Logic Programming field. Each of the three systems presented offers innovative ideas both at the theoretical and applicative level. Nevertheless, the are many more important systems and algorithm that we did not cover.

Another query-based system similar to HORN is the DUCE system due to Muggleton and McDermott [49]. The DUCE system infers propositional theories using propositional *reverse resolution* operators, i.e. the `V` and `W` operators (see Section 5.1) an the *least general generalization* (see Section 4.2). It allows for *predicate invention* and uses as heuristic the compression rate of a refined theory, i.e., the size of the theory. A first-order variant of DUCE is CIGOL [52], due to Muggleton and Buntine. GOLEM, due to Muggleton and Feng [53], is an example driven (as opposed to query-driven) system based on the *relative least general generalization* operator. GOLEM induces first order Horn theories and allows for functional symbols. It also enables the use of background knowledge. The example language is composed by ground atoms. It is known for its efficiency and thus the ability to handle large sets of examples. PROGOL is yet another system due to Muggleton [50]. This system provides a guarantee on the minimality of the size of the inferred hypothesis. Interestingly enough PROGOL was responsible for inferring useful, human understandable pieces of knowledge in the field of Biology [69]. More recent advances in ILP utilise the recent development of the answer-set programming (ASP) paradigm. More precisely, an approach called *meta-level* programming has seen fruition. In this paradigm the ILP problem, the hypothesis space, the example and a language bias is described in answer-set programming. Then, an ASP solver undertakes the tasks od solving the ILP problem [19]. Examples in this direction include Corapi's et al. ASPAL [17] and Law's et al. ILASP [39]. More information regarding meta-level programming can be found here [35].

Chapter 7

Conclusions

In this paper we surveyed main results from the literature of *inductive logic programming* (ILP). Our goal was to develop intuition about methods of symbolic learning with emphasis on the query-based approaches.

In Chapter 1 we focused on the structural properties of a hypothesis space. Particularly we focused on hypothesis spaces that are organised as *partially ordered sets* (posets) or *lattices*, by an abstract generalisation relation on the hypotheses. There we defined a learning problem abstractly, by omitting any explicit reference to a hypothesis language. We distinguished between a *deductive* problem, i.e. moving form general to specific, and an *inductive* problem, i.e. moving from specific to general. We saw that these two problems can be encoded as traversal problems in the underlying poset, or *abstract reasoning framework*. In this direction, we presented *generalisation* and *specialization* operators. Later, we introduced some query based models that assume access to an abstract oracle function that substitutes the information provided by the explicitly given examples. We also reviewed a framework for analysing learning algorithms, namely the concept of *PAC-learnability*.

In the next two chapters we discussed formalisms for our hypotheses language. Additionally we examined elementary results of deductive reasoning. In Chapter 2 we introduced the basic concepts of mathematical logic as a proposed language to describe our hypotheses. We presented *semantic* and *syntactic* entailment as a relation between logic formulas. We also established their equivalence by the *soundness* and *completeness* properties of both *propositional* and *first-order* logic. We discussed the advantages and disadvantages of these logical formalism and the trade-off between expressiveness and computational complexity. In Chapter 3 we discussed *computational logic*. We restrict our attention to a computationally, well behaved, subset of first-order logic, namely first-order Horn clauses. We presented the main properties of *resolution*, an inference operator that is sound and refutation complete. This inference rule lays at the heart of *logic programming* languages, like Prolog. We also reviewed some hypotheses of logic programming, namely the *negation as failure* and the related *closed world assumption*. We also briefly mention another paradigm based on Horn clauses, namely *answer set programming*. Both formalisms introduced in Chapters 2 and 3 constitute frameworks of deductive reasoning. Later, we examine various methods that invert deductive reasoning operators, for inductive inference.

The subsequent two chapters are devoted to inductive inference. We distinguish between single clause and multiple clause hypothesis learning. The first setting is discussed in Chapter 4. Is this setting our hypothesis space is constituted by single Horn clauses. The generality relation between the clauses is given by θ*-subsumption*. The Horn clauses will share the same predicate as head, while a clause c_1 is considered more general than c_2 when c_1 θ-subsumes c_2. θ-Subsumption constitutes a unary generalization operator, contrarily to that we also define the *least general generalization* (lgg) operator for sets of clauses. In Chapter 5 we discuss multiple clause induction, where a single hypothesis is formed by a *set* of Horn clauses. Namely, each element of the resulting hypothesis space is a set of clauses, or a logic programme. In the literature multiple clause induction is also refereed to as *theory revision*. In this setting we examine the *inverse resolution* operators, that is the `V` and `W` operators. Additionally, we review a special case of resolution inversion, namely *abduction*. Despite abduction being yet another generalization operator, it differs from the inverse resolution operators mention above intuitively, as it infers missing facts, rather than missing rules from a given theory.

In Chapter 6 we regarded implementation issues of the previously established theory. At the beginning of the chapter we present some generic algorithms that correspond to different search strategies employed by the implemented systems. We make a rough distinction between, *top-down* and *bottom-up* systems. The first group consists of algorithms that iteratively specialise an initial hypothesis in order to fit the given examples, while the latter group consist of algorithms that iteratively generalise a hypothesis beginning from the given set of examples. In addition to these groups we presented a third search method for the query-based setting. We establish that a query based method does not fall to the above two categories[1]. After we have established the algorithmic background, we review three ILP systems. The first is the First Order Inductive Learner (FOIL) [60] a top-down algorithm that iteratively learns one clause at a time. Then we proceed to two query-based approaches. The first being HORN [4] a propositional system that employs *membership* an *equivalence* queries. The HORN algorithm is of special theoretical interest, since it *exactly identifies* a propositional theory in *polynomial* query and time complexity. The other query-based system is the MODEL INFERENCE SYSTEM [66] (MODEL). The MODEL system learns first-order Horn theories and it is capable of identifying any theory *to the limit*. Despite its expressive power, MODEL needs a substantial amount of queries in order to learn even relatively simple programmes. On the other hand, the theoretical work done for this system can provide a framework for inductive reasoning. We conclude Chapter 6 by briefly discussing other inductive systems.

7.1 Meta-level Programming, Noisy Data & Neurosymbolism

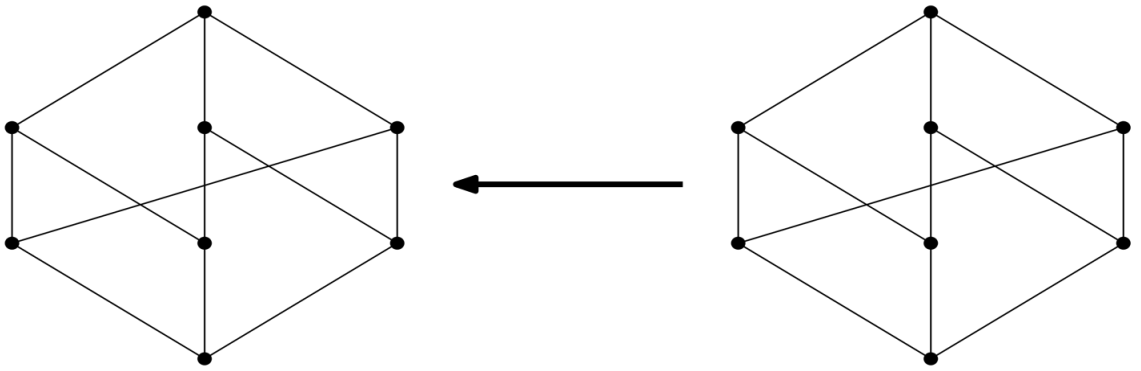
In this paper we primarily focused on the ILP techniques developed in the last decade of the previous century. The core ideas in these approaches was the representation of hypotheses in first-order Horn clauses, organising the hypothesis space with a generality relation, and traversing this space using syntactic *approximations* of logical entailment. This methodology drew its ideas from relative work from philosophers of science, trying to formalise and automate their conclusions [66]. This formal, symbolic approach to machine learning succeed in some areas that even today's cutting edge deep neural networks lack. Namely, the ILP methods regarded here are *data efficient*, a handful of examples is usually appropriate to learn simple programmes, while even for relatively complex programmes a few dozens of examples are adequate. Additionally, the knowledge learned is *transferable*. Since the acquired knowledge is in the form of Horn clauses, they can be just "copy-pasted" to any other system. Lastly, both the *process* of learning, and the *representation of knowledge* are *explainable*. The learning process is solely driven by the examples. Each revision of hypothesis can be traced back to the example, or set of examples, that caused this transition. This can be done *deterministically*, thus when a system arrives at an unwanted theory, we can determine the example that caused this divergence. Moreover, the acquired knowledge is *human readable* and *understandable*, since it is written as first-order Horn clauses. Thus, a specialist can both evaluate the learned theory and enrich its own knowledge.

On the other hand, there are some areas that classical ILP lacks compered to today's cutting edge deep neural networks methods. Traditional ILP methods, despite being data efficient, cannot handle large data sets ("big-data"). Additionally, traditional ILP requires structured input, usually in the form of relations. Thus, it is unable to handle raw data e.g. in the form of pixels or waveforms. Moreover, ILP cannot handle well noisy data, where some of the examples are mislabeled. Lastly, the inductive systems we reviewed usually need strong language bias in order to be productive. This, essentially means that a handcrafted solution template needs to be provided to the system, in order for the system to fill the blank spots. Sometimes, this results in an engineer undertaking the majority of the workload.

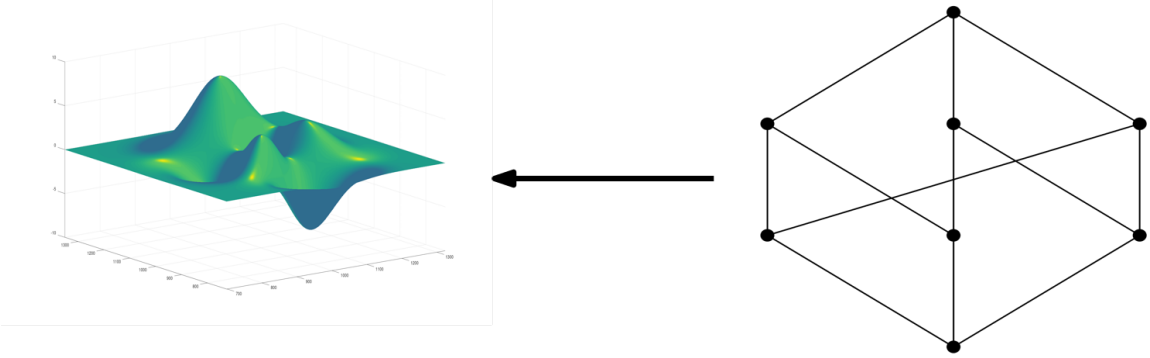
In this section we will briefly discuss some of the most modern ILP techniques that aim to overcome some of the disadvantages mentioned above. As we will see, these modern systems do not directly improve traditional ILP, rather they shift the paradigm we presented in this paper. Contrarily to traversing the hypothesis space by following a generality relation, these modern

[1] In the literature some authors may characterise a query-based system as either top-down or bottom-up, usually depending on the search strategy employed by the system in order to discover a new clause. Despite that, since a query may yield a positive or negative example, every query-based approach (known to the author) employs both specialization and generalization operators. One the other hand, usually one of these operators is simplistic, while the other is doing the heavy lifting.

methods *map* the ILP problem to alternative spaces and regard it as an optimization problem. In the first paragraph bellow we mention *meta-level programming* we the ILP problem is reduced from an induction problem to an deduction problem. This can be regarded as a form of homomorphism where a symbolic problem is mapped to a different symbolic problem (see Figure 7.1a). The latter problem is solved by ASP solvers. Subsequently we examine *neurosymbolism*. This method maps the symbolic ILP problem to a *fuzzy* (real), *continuous* space. This "real-relaxation" of the former ILP problem is then solved by a fuzzy-machine learning technique, such as a neural network (see Figure 7.1b).



(a) Meta-level programming constitutes maps the symbolic ILP problem to another symbolic space, where it treats it as a deduction problem. A model of the latter problem constitutes a solution of the induction problem.



(b) Neurosymbolism also transforms a induction problem to deduction. Moreover, it constructs a *continuous relaxation* of the resulting deduction problem which solves using continuous optimization and *sub-symbolic* machine learning techniques.

Figure 7.1: Two modern approaches to ILP. Meta-level programming and Neurosymbolism.

Meta-level programming. There is no agreed-upon definition for what meta-level ILP means, but most approaches encode the ILP problems as a meta-level logic program, i.e. a program that reasons about programs. Such meta-level approaches often delegate the search for a hypothesis to an "off-the-self" solver, after which the meta-level solution is translated back to a standard solution for the ILP problem. Meta-level approaches can often learn optimal and recursive programs. Moreover, meta-level approaches use diverse techniques and technologies. For instance, METAGOL uses a Prolog meta-interpreter to search for a proof of a meta-level Prolog program. ASPAL, ILASP translate an ILP problem into an ASP problem and use ASP solvers to find a model of the problem. All these methods utilise very different algorithms. Note that these meta-level approaches diverge from the standard *clause refinement* approach of traditional ILP [19]. We refer the interested reader about meta-level programming to. Additionally, an overview of *conflict-driven* ILP, adopted by the systems ILASP_3 and POPPER.

Since the whole ILP problem is encoded in a formal language, it enables a systematic search for the proper language bias, infeasible in traditional ILP, where language bias must be stated explicitly. Additionally, since both the original problem and the meta-problem are symbolic, the

transition back to the original space can be done without loss of information. Nevertheless, moving one level of abstraction higher comes at a computational cost. For instance, due to the limitations of popular ASP systems (e.g. grounding), it becomes increasingly difficult to scale the method to larger domains. Lastly, it seems that this higher level of abstraction obscures the explainability property we discussed at the beginning of this chapter, with respect to the *learning process*.

Neurosymbolism. Neurosymbolism essentially advances meta-level programming one step further. After the induction problem gets mapped to a deduction problem, the latter is further relaxed by mapping it to a continuous space. The motivation behind this technique is to be able to handle noisy and mislabeled data. The theoretical foundation behind this work lays in the fact the *neural networks are Turing complete*, which in return is a result of neural networks being *universal approximators*. A key contribution of neurosymbolic methods are techniques of *continuous resolution*.

We briefly discuss the differentiable-ILP (∂ILP) [24] system as a representative method. There all possible clauses (or logic programmes) are generated according to a user defined language bias. The entries of the underlying neural network's weight matrix correspond to a belief value assigned to any rule, from the interval $[0, 1]$. After the training concludes, the weights will result to a probability distribution of the set of possible rules.

The ∂ILP system and other neurosymbolic approaches achieve to expand the realm to ILP learnable tasks by allowing the system to introduce *"fuzzyness"* to the learned theory, thus being able to handle noise. Additionally, this fuzzy approach enables a system to harness the computational power utilised by deep neural network techniques, e.g. GPUs. Also, the resulting hypothesis is human readable, contrarily to most deep neural network methods. On the other hand, it is not trivial to translate the solution from the continuous space back to the symbolic space. For instance, we cannot in general translate the distribution on the space of hypotheses to an equivalent deterministic set of clauses in the same space. Lastly, the comment we made for meta-level programming about the lack of an explainable procedure also applies here. Moreover, this method also lacks in the explainability of the represented knowledge, since the fuzziness introduced to the first-order clauses constitutes an unexplainable element. To illustrate the latter fact we give the following example.

Example 8. *Assume that we want to learn the concept* `mother`$/1, $`father`$/1$ *from instances of the relations* `mother`$/1, $`father`$/1, $`parent`$/1, $`female`$/1, $`male`$/1$. *Moreover, we assume that in our dataset the examples for* `male`$/1, $`female`$/1$ *are evenly distributed, namely half of our examples are men, while the other half are women. A traditional ILP system (e.g. FOIL) will learn the following definitions.*

$$\texttt{mother(X)} \leftarrow \texttt{parent(X)}, \texttt{female(X)}. \tag{7.1}$$
$$\texttt{father(X)} \leftarrow \texttt{parent(X)}, \texttt{male(X)}. \tag{7.2}$$

Assume, now, that we can assign a probability distribution on each rule. The inference system we use will respect this probability and choose each rule randomly according to the assigned probability. This "toy" fuzzy system[2] *may come up with the definitions:*

$$p_1 = \frac{1}{2}: \quad \texttt{mother(X)} \leftarrow \texttt{parent(X)}. \tag{7.3}$$
$$p_2 = \frac{1}{2}: \quad \texttt{father(X)} \leftarrow \texttt{parent(X)}. \tag{7.4}$$

The definitions (7.3), (7.4) are, naturally statistically correct. On the other hand, we can argue that value of information and understandability provided by the definitions (7.1), (7.2) is higher[3].

7.2 Discussion & Open Problems

We close this paper with a discussion about topics for future research. Our main scientific questions have been presented throughout in this paper in Conjectures 1, 2, 3. Our main questions regard the

[2] Which does not diverge much from neurosymbolic systems such as ∂ILP.
[3] The authors of [24] address this fact by evaluating the *entropy* of the generated rules as a measure of randomness.

theoretical properties of the abstract reasoning frameworks presented here. An abstract reasoning framework is essentially a partial order and perceives reasoning problems as traversal problems in a poset and more specifically in a lattice. Our goal for the introduction of this notion was to provide a unifying framework that highlights the algorithmic properties of any inductive system. We focus on query-based systems since it might be possible to utilise the recent work on posets [20, 13, 9, 5] to provide rigorous bounds for inductive systems. Related work has been done in [3, 1, 2, 66]. The authors there consider a query based system with respect to *equivalence* and *membership* queries. Our proposed model considered *comparison* queries. The relationship between the two models remain open (see Conjecture 3). Additionally, lower bounds for the complexity, both computational and with respect to examples, are presented in [16], but are based on the theory of PAC-learnability [73]. Comparing the analysis in [16] with a query-based analysis with respect to a comparison setting presented in [20] will pose an interesting problem. The former analysis in [16], since it is based on PAC-learnability, is a form of *probabilistic* analysis. The latter setting in [20] is a time of *amortized* analysis that does not rely on probabilistic assumptions. Another unifying formulation for the ILP problems is presented in [63]. This formulation is based on the *cover* relation, contrarily to our proposal which is based on the generality relation. Naturally, the relation between the two formalisms is open. If Conjecture 3 proved to be correct it will enable us to apply the theoretical bounds on sorting and selection problems on posets to induction. This will constitute a step forward to provide theoretical guarantees for *data-efficiency*.

7.2.1 On Explainability

The reason that we insist on the importance of a generality based framework is that we believe that it could provide a formalism to study *explainability*. The issue of explainability of inductive algorithms consist one of the great open questions in machine learning (ML) today [42]. Nevertheless a rigorous definition of the problem is yet to be formulated. We differentiate between two aspects of the problem, highlighted at the beginning of this chapter.

- *Human understandable knowledge representation.* Here we insist on the readability of the knowledge generated by the learning algorithm. A informal model in this direction is presented in [45], known as *ultra-strong ML*. There a learned hypothesis is expected to not only be accurate but to also demonstrably improve the performance of a human when provided with the learned hypothesis.

- *Invertible deterministic learning process.* Here we insist that the learning algorithm is deterministically invertible. In other words, for each transition from a hypothesis h_1 to h_2 there must be a set of examples E that are deterministically responsible for this transition. This way the learning process becomes "debugable", each update of the current hypothesis can be traced back to the set of examples responsible for this update.

The first notion of explainability above is still "fuzzy" and formalizing it would be a non trivial matter. In other words, it may be argued that a notion of "human understandable" is subjective. We could argue that a probability distribution is less readable than a theory presented in logical formulas. On the other hand, a lengthy logic theory with multiple complex clauses can be quite difficult to understand [65]. Contrarily to this approach we argue that we could characterise a ML system explainable if it is, essentially, "debugable ". In such a system we could answer "why" each hypothesis is learned, regardless of the representation of the generated knowledge. Essentially, we demand that the system provides a "proof" of the learned hypothesis, as a sequence of example sets that resulted in this hypothesis. In this approach the notion of explainability coincides with *determinism*. Following this perspective, we could construct proofs about the explainability of a system. Notice that intuitively traditional ILP systems should be "explainable", with respect to the proposed notion, since these systems essentially invert logic proofs.

We believe that the above proposition for the notion of explainability can be extended outside of the realm of symbolic methods. For instance, a hypothesis of a linear classifier in the d-dimensional real space is a hyperplane in that space. We could organise the set of hypotheses with a generality relation, forming an abstract reasoning framework. Therefore, we could analyse that system for explainability, with respect to the *"invertible deterministic learning"* criterion presented above. This could provide an alternate route for merging symbolic and fuzzy methods, by mapping real functions (the hypotheses) to a discrete lattice organised by generality. Therefore inverting the direction (see Figure 7.2) of the reduction presented in the previous section.

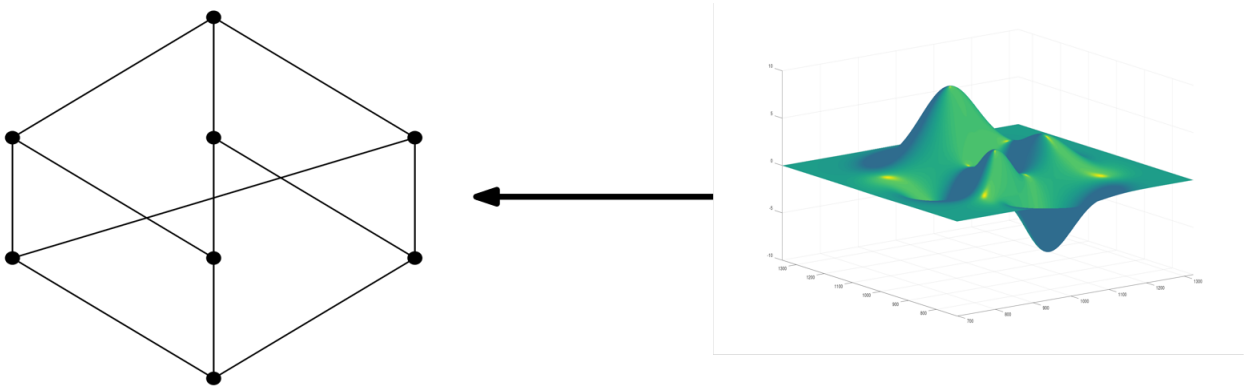


Figure 7.2: In a fuzzy (real) machine learning problem the hypotheses are usually denoted as real functions in a multidimensional space. We could organise the set of hypotheses by a generality relation, thus mapping it to a (possible infinite) lattice. Then, it would be possible to test the algorithm for explainability, with respect to the "invertible deterministic learning" criterion.

Bibliography

[1] Dana Angluin. "Learning With Hints". In: *Proceedings of the First Annual Workshop on Computational Learning Theory, COLT '88, Cambridge, MA, USA, August 3-5, 1988*. Ed. by David Haussler and Leonard Pitt. ACM/MIT, 1988, pp. 167–181. URL: http://dl.acm.org/citation.cfm?id=93075.

[2] Dana Angluin. "Negative Results for Equivalence Queries". In: *Mach. Learn.* 5 (1990), pp. 121–150. DOI: 10.1007/BF00116034. URL: https://doi.org/10.1007/BF00116034.

[3] Dana Angluin. "Queries revisited". In: *Theor. Comput. Sci.* 313.2 (2004), pp. 175–194. DOI: 10.1016/J.TCS.2003.11.004. URL: https://doi.org/10.1016/j.tcs.2003.11.004.

[4] Dana Angluin, Michael Frazier, and Leonard Pitt. "Learning Conjunctions of Horn Clauses". In: *Mach. Learn.* 9 (1992), pp. 147–164. DOI: 10.1007/BF00992675. URL: https://doi.org/10.1007/BF00992675.

[5] Indranil Banerjee and Dana S. Richards. "Sorting Under Forbidden Comparisons". In: *15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22-24, 2016, Reykjavik, Iceland*. Ed. by Rasmus Pagh. Vol. 53. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 22:1–22:13. DOI: 10.4230/LIPIcs.SWAT.2016.22. URL: https://doi.org/10.4230/LIPIcs.SWAT.2016.22.

[6] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2010. ISBN: 978-0-521-14775-0. URL: http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/knowledge-representation-reasoning-and-declarative-problem-solving.

[7] Mark de Berg et al. *Computational geometry: algorithms and applications, 3rd Edition*. Springer, 2008. ISBN: 9783540779735. URL: https://www.worldcat.org/oclc/227584184.

[8] Francesco Bergadano and Daniele Gunetti. *Inductive logic programming - from machine learning to software engineering*. MIT Press, 1996. ISBN: 978-0-262-02393-1.

[9] Arindam Biswas, Varunkumar Jayapaul, and Venkatesh Raman. "Improved Bounds for Poset Sorting in the Forbidden-Comparison Regime". In: *Algorithms and Discrete Applied Mathematics - Third International Conference, CALDAM 2017, Sancoale, Goa, India, February 16-18, 2017, Proceedings*. Ed. by Daya Ram Gaur and N. S. Narayanaswamy. Vol. 10156. Lecture Notes in Computer Science. Springer, 2017, pp. 50–59. DOI: 10.1007/978-3-319-53007-9_5. URL: https://doi.org/10.1007/978-3-319-53007-9%5C_5.

[10] Wray L. Buntine. "Generalized Subsumption and Its Applications to Induction and Redundancy". In: *Artif. Intell.* 36.2 (1988), pp. 149–176. DOI: 10.1016/0004-3702(88)90001-X. URL: https://doi.org/10.1016/0004-3702(88)90001-X.

[11] Wray L. Buntine. "Induction of Horn Clauses: Methods and the Plausible Generalization Algorithm". In: *Int. J. Man Mach. Stud.* 26.4 (1987), pp. 499–519. DOI: 10.1016/S0020-7373(87)80084-6. URL: https://doi.org/10.1016/S0020-7373(87)80084-6.

[12] Pedro Cabalar. "Answer Set; Programming?" In: *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*. Ed. by Marcello Balduccini and Tran Cao Son. Vol. 6565. Lecture Notes in Computer Science. Springer, 2011, pp. 334–343. DOI: 10.1007/978-3-642-20832-4_21. URL: https://doi.org/10.1007/978-3-642-20832-4%5C_21.

[13] Jean Cardinal and Samuel Fiorini. "On Generalized Comparison-Based Sorting Problems". In: *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*. Ed. by Andrej Brodnik et al. Vol. 8066. Lecture Notes in Computer Science. Springer, 2013, pp. 164–175. DOI: 10.1007/978-3-642-40273-9_12. URL: https://doi.org/10.1007/978-3-642-40273-9%5C_12.

[14] Stefano Ceri, Georg Gottlob, and Letizia Tanca. "What you Always Wanted to Know About Datalog (And Never Dared to Ask)". In: *IEEE Trans. Knowl. Data Eng.* 1.1 (1989), pp. 146–166. DOI: 10.1109/69.43410. URL: https://doi.org/10.1109/69.43410.

[15] Peter Clark and Tim Niblett. "The CN2 Induction Algorithm". In: *Mach. Learn.* 3 (1989), pp. 261–283. DOI: 10.1007/BF00116835. URL: https://doi.org/10.1007/BF00116835.

[16] William W. Cohen and C. David Page Jr. "Polynomial Learnability and Inductive Logic Programming: Methods and Results". In: *New Gener. Comput.* 13.3&4 (1995), pp. 369–409. DOI: 10.1007/BF03037231. URL: https://doi.org/10.1007/BF03037231.

[17] Domenico Corapi, Alessandra Russo, and Emil Lupu. "Inductive Logic Programming in Answer Set Programming". In: *Inductive Logic Programming - 21st International Conference, ILP 2011, Windsor Great Park, UK, July 31 - August 3, 2011, Revised Selected Papers*. Ed. by Stephen H. Muggleton, Alireza Tamaddoni-Nezhad, and Francesca A. Lisi. Vol. 7207. Lecture Notes in Computer Science. Springer, 2011, pp. 91–97. DOI: 10.1007/978-3-642-31951-8_12. URL: https://doi.org/10.1007/978-3-642-31951-8%5C_12.

[18] Thomas M. Cover and Joy A. Thomas. *Elements of information theory (2. ed.)* Wiley, 2006. ISBN: 978-0-471-24195-9. URL: http://www.elementsofinformationtheory.com/.

[19] Andrew Cropper and Sebastijan Dumancic. "Inductive Logic Programming At 30: A New Introduction". In: *J. Artif. Intell. Res.* 74 (2022), pp. 765–850. DOI: 10.1613/JAIR.1.13507. URL: https://doi.org/10.1613/jair.1.13507.

[20] Constantinos Daskalakis et al. "Sorting and Selection in Posets". In: *SIAM J. Comput.* 40.3 (2011), pp. 597–622. DOI: 10.1137/070697720. URL: https://doi.org/10.1137/070697720.

[21] R. P. Dilworth. "A Decomposition Theorem for Partially Ordered Sets". In: *Annals of Mathematics* 51.1 (1950), pp. 161–166. ISSN: 0003486X. URL: http://www.jstor.org/stable/1969503 (visited on 04/17/2022).

[22] D.L. Dowe. *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence: Papers from the Ray Solomonoff 85th Memorial Conference, Melbourne, VIC, Australia, November 30 – December 2, 2011*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013. ISBN: 9783642449581. URL: https://books.google.gr/books?id=P2O5BQAAQBAJ.

[23] Herbert B. Enderton. *A mathematical introduction to logic*. Academic Press, 1972. ISBN: 978-0-12-238450-9.

[24] Richard Evans and Edward Grefenstette. "Learning Explanatory Rules from Noisy Data". In: *J. Artif. Intell. Res.* 61 (2018), pp. 1–64. DOI: 10.1613/JAIR.5714. URL: https://doi.org/10.1613/jair.5714.

[25] Giorgos Flouris. "On Belief Change and Ontology Evolution". https://elocus.lib.uoc.gr/dlib/f/f/5/metadata-dlib-2006flouris.tkl. PhD thesis. University of Crete, 2006. URL: https://elocus.lib.uoc.gr/dlib/f/f/5/metadata-dlib-2006flouris.tkl.

[26] Michael Frazier and Leonard Pitt. "Learning From Entailment: An Application to Propositional Horn Sentences". In: *Machine Learning, Proceedings of the Tenth International Conference, University of Massachusetts, Amherst, MA, USA, June 27-29, 1993*. Ed. by Paul E. Utgoff. Morgan Kaufmann, 1993, pp. 120–127. DOI: 10.1016/B978-1-55860-307-3.50022-8. URL: https://doi.org/10.1016/b978-1-55860-307-3.50022-8.

[27] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990. ISBN: 0716710455.

[28] MARTIN GEBSER et al. "Multi-shot ASP solving with clingo". In: *Theory and Practice of Logic Programming* 19.1 (2019), pp. 27–82. DOI: 10.1017/S1471068418000054.

[29] Goran Gogic et al. "The Comparative Linguistics of Knowledge Representation". In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*. Morgan Kaufmann, 1995, pp. 862–869. URL: http://ijcai.org/Proceedings/95-1/Papers/111.pdf.

[30] E Mark Gold. "Language identification in the limit". In: *Information and Control* 10.5 (1967), pp. 447–474. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(67)91165-5. URL: https://www.sciencedirect.com/science/article/pii/S0019995867911655.

[31] Peter Grünwald and Paul M. B. Vitányi. "Shannon Information and Kolmogorov Complexity". In: *CoRR* cs.IT/0410002 (2004). URL: http://arxiv.org/abs/cs.IT/0410002.

[32] Peter D Grünwald, In Jae Myung, and Mark A Pitt. *Advances in minimum description length: Theory and applications*. MIT press, 2005.

[33] Frank van Harmelen, Vladimir Lifschitz, and Bruce W. Porter, eds. *Handbook of Knowledge Representation*. Vol. 3. Foundations of Artificial Intelligence. Elsevier, 2008. ISBN: 978-0-444-52211-5. URL: https://www.sciencedirect.com/science/bookseries/15746526/3.

[34] C. Howson. *Logic with Trees: An Introduction to Symbolic Logic*. Logic with Trees: An Introduction to Symbolic Logic. Routledge, 1997. ISBN: 9780415133425. URL: https://books.google.gr/books?id=A15fsL--IKQC.

[35] Katsumi Inoue. "Meta-Level Abduction". In: *FLAP* 3.1 (2016), pp. 7–36. URL: http://www.collegepublications.co.uk/downloads/ifcolog00005.pdf.

[36] Christian Jäkel and Stefan E. Schmidt. "Optimization problems on posets with regard to formal concept analysis". In: *Int. J. Approx. Reason.* 142 (2022), pp. 196–205. DOI: 10.1016/J.IJAR.2021.12.005. URL: https://doi.org/10.1016/j.ijar.2021.12.005.

[37] Varunkumar Jayapaul. "Sorting and Selection in Restriction Model". https://libarchive.cmi.ac.in/theses/varunkumarj_cs2017.pdf. PhD thesis. Chennai Mathematical Institute, 2017. URL: https://libarchive.cmi.ac.in/theses/varunkumarj_cs2017.pdf.

[38] Robert A. Kowalski. "The Early Years of Logic Programming". In: *Commun. ACM* 31.1 (1988), pp. 38–43. DOI: 10.1145/35043.35046. URL: https://doi.org/10.1145/35043.35046.

[39] Mark Law, Alessandra Russo, and Krysia Broda. "Inductive Learning of Answer Set Programs". In: *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*. Ed. by Eduardo Fermé and João Leite. Vol. 8761. Lecture Notes in Computer Science. Springer, 2014, pp. 311–325. DOI: 10.1007/978-3-319-11558-0_22. URL: https://doi.org/10.1007/978-3-319-11558-0%5C_22.

[40] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the theory of computation, 2nd Edition*. Prentice Hall, 1998. ISBN: 978-0-13-262478-7.

[41] Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications, 4th Edition*. Texts in Computer Science. Springer, 2019. ISBN: 978-3-030-11297-4. DOI: 10.1007/978-3-030-11298-1. URL: https://doi.org/10.1007/978-3-030-11298-1.

[42] Ricards Marcinkevics and Julia E. Vogt. "Interpretable and explainable machine learning: A methods-centric overview with concrete examples". In: *WIREs Data. Mining. Knowl. Discov.* 13.3 (2023). DOI: 10.1002/WIDM.1493. URL: https://doi.org/10.1002/widm.1493.

[43] Jerzy Marcinkowski. "A Horn Clause that Implies and Undecidable Set of Horn Clauses". In: *Computer Science Logic, 7th Workshop, CSL '93, Swansea, United Kingdom, September 13-17, 1993, Selected Papers*. Ed. by Egon Börger, Yuri Gurevich, and Karl Meinke. Vol. 832. Lecture Notes in Computer Science. Springer, 1993, pp. 223–237. DOI: 10.1007/BFB0049334. URL: https://doi.org/10.1007/BFb0049334.

[44] Ryszard S. Michalski. "A Theory and Methodology of Inductive Learning". In: *Artif. Intell.* 20.2 (1983), pp. 111–161. DOI: 10.1016/0004-3702(83)90016-4. URL: https://doi.org/10.1016/0004-3702(83)90016-4.

[45] Donald Michie. "Machine Learning in the Next Five Years". In: *Proceedings of the Third European Working Session on Learning, EWSL 1988, Turing Institute, Glasgow, UK, October 3-5, 1988*. Ed. by Derek H. Sleeman. Pitman Publishing, 1988, pp. 107–122.

[46] L. Mirsky. "A Dual of Dilworth's Decomposition Theorem". In: *The American Mathematical Monthly* 78.8 (1971), pp. 876–877. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/2316481 (visited on 04/17/2022).

[47] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. ISBN: 978-0-07-042807-2. URL: https://www.worldcat.org/oclc/61321007.

[48] S. Muggleton. *Inductive Logic Programming*. A.P.I.C. studies in data processing. Academic Press, 1992. ISBN: 9780125097154. URL: https://books.google.gr/books?id=IYAewM0zvTYC.

[49] Stephen H. Muggleton. "Duce, An Oracle-based Approach to Constructive Induction". In: *Proceedings of the 10th International Joint Conference on Artificial Intelligence. Milan, Italy, August 23-28, 1987*. Ed. by John P. McDermott. Morgan Kaufmann, 1987, pp. 287–292. URL: http://ijcai.org/Proceedings/87-1/Papers/059.pdf.

[50] Stephen H. Muggleton. "Inverting Entailment and Progol". In: *Machine Intelligence 14, Proceedings of the Fourteenth Machine Intelligence Workshop, held at Hitachi Advanced Research Laboratories, Tokyo, Japan, November 1993*. Ed. by Koichi Furukawa, Donald Michie, and Stephen H. Muggleton. Oxford University Press, 1993, pp. 135–190.

[51] Stephen H. Muggleton and Wray L. Buntine. "Machine Invention of First Order Predicates by Inverting Resolution". In: *Machine Learning, Proceedings of the Fifth International Conference on Machine Learning, Ann Arbor, Michigan, USA, June 12-14, 1988*. Ed. by John E. Laird. Morgan Kaufmann, 1988, pp. 339–352.

[52] Stephen H. Muggleton and Wray L. Buntine. "Machine Invention of First Order Predicates by Inverting Resolution". In: *Machine Learning, Proceedings of the Fifth International Conference on Machine Learning, Ann Arbor, Michigan, USA, June 12-14, 1988*. Ed. by John E. Laird. Morgan Kaufmann, 1988, pp. 339–352.

[53] Stephen H. Muggleton and Cao Feng. "Efficient Induction of Logic Programs". In: *Algorithmic Learning Theory, First International Workshop, ALT '90, Tokyo, Japan, October 8-10, 1990, Proceedings*. Ed. by Setsuo Arikawa et al. Springer/Ohmsha, 1990, pp. 368–381.

[54] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. *Foundations of Inductive Logic Programming*. Vol. 1228. Lecture Notes in Computer Science. Springer, 1997. ISBN: 3-540-62927-0. DOI: 10.1007/3-540-62927-0. URL: https://doi.org/10.1007/3-540-62927-0.

[55] Christos H. Papadimitriou. "The Complexity of Knowledge Representation". In: *Proceedings of the Eleventh Annual IEEE Conference on Computational Complexity, Philadelphia, Pennsylvania, USA, May 24-27, 1996*. Ed. by Steven Homer and Jin-Yi Cai. IEEE Computer Society, 1996, pp. 244–248. DOI: 10.1109/CCC.1996.507686. URL: https://doi.org/10.1109/CCC.1996.507686.

[56] Merkouris Papamichail. "Sorting and Selection Problems in Partially Ordered Sets". https://pergamos.lib.uoa.gr/uoa/dl/object/3232651. Master's Thesis. National and Kapodistrian University of Athens, 2022. URL: https://pergamos.lib.uoa.gr/uoa/dl/object/3232651.

[57] Gordon D Plotkin. "A note on inductive generalization". In: vol. 5. 1. 1970, pp. 153–163.

[58] Gordon D. Plotkin. "A Further Note on Inductive Generalization". In: 1971. URL: https://api.semanticscholar.org/CorpusID:8230949.

[59] J. Ross Quinlan. "Induction of Decision Trees". In: *Mach. Learn.* 1.1 (1986), pp. 81–106. DOI: 10.1023/A:1022643204877. URL: https://doi.org/10.1023/A:1022643204877.

[60] J. Ross Quinlan. "Learning Logical Definitions from Relations". In: *Mach. Learn.* 5 (1990), pp. 239–266. DOI: 10.1007/BF00117105. URL: https://doi.org/10.1007/BF00117105.

[61] Luc De Raedt. *Logical and relational learning*. Cognitive Technologies. Springer, 2008. ISBN: 978-3-540-20040-6. DOI: 10.1007/978-3-540-68856-3. URL: https://doi.org/10.1007/978-3-540-68856-3.

[62] Luc De Raedt and Saso Dzeroski. "First-Order jk-Clausal Theories are PAC-Learnable". In: *Artif. Intell.* 70.1-2 (1994), pp. 375–392. DOI: 10.1016/0004-3702(94)90112-0. URL: https://doi.org/10.1016/0004-3702(94)90112-0.

[63] Luc De Raedt and Kristian Kersting. "Probabilistic Inductive Logic Programming". In: *Probabilistic Inductive Logic Programming - Theory and Applications*. Ed. by Luc De Raedt et al. Vol. 4911. Lecture Notes in Computer Science. Springer, 2008, pp. 1–27. DOI: 10.1007/978-3-540-78652-8_1. URL: https://doi.org/10.1007/978-3-540-78652-8%5C_1.

[64] Joachim Schimpf and Kish Shen. "ECL^{i}PS^{e} - From LP to CLP". In: *Theory Pract. Log. Program.* 12.1-2 (2012), pp. 127–156. DOI: 10.1017/S1471068411000469. URL: https://doi.org/10.1017/S1471068411000469.

[65] Ute Schmid et al. "How Does Predicate Invention Affect Human Comprehensibility?" In: *Inductive Logic Programming - 26th International Conference, ILP 2016, London, UK, September 4-6, 2016, Revised Selected Papers*. Ed. by James Cussens and Alessandra Russo. Vol. 10326. Lecture Notes in Computer Science. Springer, 2016, pp. 52–67. DOI: 10.1007/978-3-319-63342-8_5. URL: https://doi.org/10.1007/978-3-319-63342-8%5C_5.

[66] Ehud Shapiro. "Inductive Inference of Theories from Facts". In: *Computational Logic - Essays in Honor of Alan Robinson*. Ed. by Jean-Louis Lassez and Gordon D. Plotkin. The MIT Press, 1991, pp. 199–254.

[67] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997. ISBN: 978-0-534-94728-6.

[68] E. Sperner. "Ein Satz über Untermengen einer endlichen Menge". In: *Mathematische Zeitschrift* 27 (1928), pp. 544–548. URL: http://eudml.org/doc/167993.

[69] Ashwin Srinivasan et al. "Theories for Mutagenicity: A Study in First-Order and Feature-Based Induction". In: *Artif. Intell.* 85.1-2 (1996), pp. 277–299. DOI: 10.1016/0004-3702(95)00122-0. URL: https://doi.org/10.1016/0004-3702(95)00122-0.

[70] Tommi Syrjänen. "Omega-Restricted Logic Programs". In: *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings*. Ed. by Thomas Eiter, Wolfgang Faber, and Miroslaw Truszczynski. Vol. 2173. Lecture Notes in Computer Science. Springer, 2001, pp. 267–279. DOI: 10.1007/3-540-45402-0_20. URL: https://doi.org/10.1007/3-540-45402-0%5C_20.

[71] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern recognition*. Academic Press, 1999. ISBN: 978-0-12-686140-2.

[72] W.T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory*. Johns Hopkins Studies in Nineteenth C Architecture Series. Johns Hopkins University Press, 1992. ISBN: 9780801869778. URL: https://books.google.gr/books?id=VAEgDWgFZ10C.

[73] Leslie G. Valiant. "A Theory of the Learnable". In: *Commun. ACM* 27.11 (1984), pp. 1134–1142. DOI: 10.1145/1968.1972. URL: https://doi.org/10.1145/1968.1972.

[74] Philip Wadler and Avaya Labs. "Proofs are Programs: 19th Century Logic and 21st Century Computing". In: 2000. URL: https://api.semanticscholar.org/CorpusID:62063863.

[75] Jan Wielemaker et al. "SWI-Prolog". In: *Theory and Practice of Logic Programming* 12.1-2 (2012), pp. 67–96. ISSN: 1471-0684.