Second-order inductive learning

Flach, P.A.

Publication date:
1990

Document Version
Publisher's PDF, also known as Version of record

ITK

RESEARCH
REPORT

INSTITUTE FOR LANGUAGE TECHNOLOGY AND ARTIFICIAL INTELLIGENCE

INSTITUUT VOOR TAAL- EN KENNISTECHNOLOGIE

ITK Research Report No. 10

January 1990

Second-order inductive learning

Peter A. Flach

A preliminary version of this paper appeared in
Analogical and Inductive Inference AII'89
K.P. Jantke (ed.), Lecture Notes in Computer Science 397,
Springer Verlag, Berlin, 1989, pp. 202-216.

ISSN 0924-7807

Institute for Language Technology and Artificial Intelligence,
Tilburg University, The Netherlands

Second-order inductive learning

Peter A. Flach

ABSTRACT

In this paper, we present a new paradigm for inductive learning, called *second-order inductive learning*. It differs from concept learning from examples in that examples are not instances of the hypothesis to be learned, but rather instances of a *prototype* (i.e., a typical member of the extension) of the hypothesis to be learned. The paradigm is introduced by means of an example problem from the field of conceptual modeling. We analyse the reasons why a naive solution to that problem is not fully satisfactory by studying the *Version Space model*. Once it is clear why this model is not directly applicable, we attempt to restore it by defining the notion of a *Generalised Version Space model*. An alternative formulation of the problem is given in terms of logic.

Contents

1. Introduction

Inductive learning is an important subject in Artificial Intelligence research, because it has many practical applications, and because it can be sufficiently formalised. Examples of such formalisations are [Mitchell 1982] and [Laird 1988]. When devising a formal model of inductive learning, it is important to keep the model as general as possible, thereby indicating how more specific models can be obtained from the general model by making specific choices for, e.g., the structure of the hypothesis space involved. Within the field of discrete mathematics, the theory of lattices provides a particularly good example of such a general model.

It is in a general model for inductive learning that we are interested. However, it is difficult to build such an abstract model from scratch. To avoid this difficulty, many existing models are based on specific learning situations. For instance, Mitchell introduces his Version Spaces by referring to the problem of concept learning from examples, where an example is taken to be a member from the class described by the concept. The drawback of such an intuitively appealing model is, that some basic assumptions are left implicit. To make these assumptions explicit, we have been looking for learning situations for which Version Spaces are not adequate. One such learning situation is described in this paper.

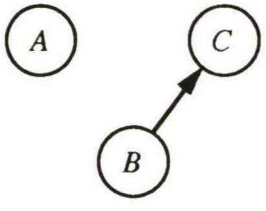
This learning situation, which we call the Schema Inference Problem, will be our guideline throughout the paper. We will attempt to give learning algorithms for that problem, and in doing so we will describe a learning paradigm called *second-order inductive learning* in which the problem fits. This learning paradigm differs from existing models in various respects. While this is our main objective, we also point at fundamental implicit assumptions underlying the Version Space model when we encounter them. Such assumptions concern properties of a model for inductive learning, and are therefore 'meta-theoretical' notions. In this respect, our efforts can be interpreted as being directed 'towards a *meta-theory* of inductive learning' [Flach 1990a].

2. The Schema Inference Problem

2.1 The problem

The research reported here originates from the following problem: given the (partial) contents of a knowledge base, derive (parts of) the conceptual model of that knowledge base. This is clearly an inductive problem, and it will be studied here from the viewpoint of inductive learning. Aspects of this problem, restricted to finding functional and multivalued dependencies for a relational database, will be discussed in a forthcoming report [Flach 1990b]. Here, we study the problem of inducing type hierarchies from type assignments to individuals, which we call the Schema Inference Problem. The significance of this problem from the standpoint of conceptual modeling is illustrated by, e.g., [Vermeir & Nijssen 1982]. The use of inductive methods for these problems has, to the best of our knowledge, not been proposed before.

SCHEMA INFERENCE PROBLEM. (*syntax*) Lowercase letters $a, b, c, \ldots$ are *constant symbols* denoting individuals; uppercase letters $A, B, C, \ldots$ are *type symbols* denoting types. A *type set* is a set of type symbols. A *schema sentence* is a statement of the form $A \rightarrow B$. A *schema* Σ over a type set σ is a set of schema sentences containing only type symbols from σ; σ is called the *domain* of Σ. For convenience, we adopt a graphical representation of schemas as follows: each type symbol in σ is represented by a distinct circle with the type symbol written inside, and for every schema sentence $A \rightarrow B$ in the schema there is a directed arrow from the circle representing A to the circle representing B. E.g., the schema $\{B \rightarrow C\}$ over $\{A,B,C\}$ is represented as[1]



(*semantics*) A *type population* is a set of constant symbols. A *schema population* for a schema Σ with domain σ is a function Π mapping type symbols in σ to type populations such that for each two type symbols A and B in σ, $A \rightarrow_\Sigma B$ implies $\Pi(A) \subseteq \Pi(B)$; $\rightarrow_\Sigma$ is defined to be the transitive closure of $\rightarrow$ (interpreted as a relation between type symbols, defined by the schema Σ).

(*examples*) A *positive example* is a statement of the form $A(a)$; a *negative example* is a statement of the form $\neg B(c)$.

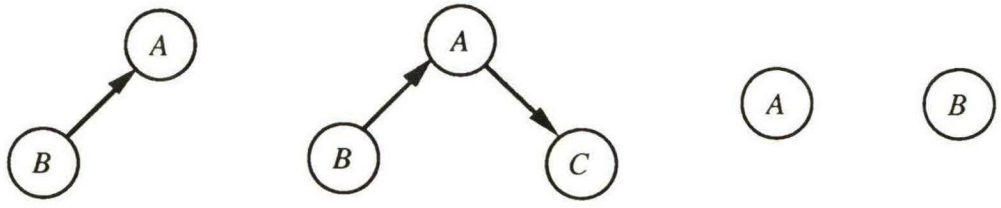
(*consistency*) A schema is *consistent* with a set of positive and negative examples, iff there is a schema population Π such that:

- for each positive example $A(a)$, $a \in \Pi(A)$;
- for each negative example $\neg B(c)$, $c \notin \Pi(B)$.

[1] Note, that this is according to the definitions: a schema need not contain all type symbols in its domain. In the corresponding diagram, the type symbols that are not contained in the schema are represented by circles not connected by arrows to any other circle.

(*learning*) The problem is, to find a **learning algorithm** that, given a domain σ for a schema, takes one example at a time, and after each example outputs a schema over σ that is consistent with the examples seen so far. ■

Intuitively, a positive example $A(a)$ states that individual a is of type A; likewise, a negative example $\neg B(c)$ states that individual c is not of type B. A schema sentence $A \rightarrow B$ states that type A is a subtype of type B. Suppose we have the positive examples $A(a)$, $A(b)$ and $B(b)$, and the negative example $\neg B(a)$, then the schema $\{B \rightarrow A\}$ (fig. 1.a) is consistent with these examples, witnessed by the schema population Π given by $\Pi(A)=\{a,b\}$ and $\Pi(B)=\{b\}$. Of course, there are many more schemas that are consistent with the examples, for instance the schema $\{B \rightarrow A, A \rightarrow C\}$ (fig. 1.b), or the empty schema $\varnothing$ over $\{A,B\}$ (fig. 1.c). Note however, that no consistent schema can contain the schema sentence $A \rightarrow B$, because of the examples $A(a)$ and $\neg B(a)$.



a. $\{B \rightarrow A\}$ *b.* $\{B \rightarrow A, A \rightarrow C\}$ *c.* $\varnothing$

Figure 1. Some schemas consistent with the examples $A(a)$, $A(b)$, $B(b)$ and $\neg B(a)$.

It seems advisable not to add any type symbols to a schema that are not present in any example (as happened in fig. 1.b). However, there is one exception to this rule: we might want to express that two types A and B have a common subtype X not equal to A or B. For instance, given the positive examples $A(a)$, $A(b)$, $B(b)$ and $B(c)$ (and perhaps the negative examples $\neg A(c)$ and $\neg B(a)$), it might be hypothesised that $\{X \rightarrow A, X \rightarrow B\}$ is the intended schema with population $\Pi(A)=\{a,b\}$, $\Pi(B)=\{b,c\}$, $\Pi(X)=\{b\}$. Put differently, by introducing an auxiliary type X, we can express that types A and B have something in common. Still, we want to say that the domain of the resulting schema is $\{A,B\}$. Therefore, we extend the definition of a schema as follows: a schema Σ over a type set σ is a set of schema sentences; the type symbols occurring in Σ but not in σ denote *auxiliary types*, and each auxiliary type symbol should occur on the left hand side of at least two schema sentences in Σ. The definition of a schema population is left unchanged, that is, the domain of Π is the type set σ. Populations for auxiliary types X can then be derived by taking the intersection of all types A for which the schema contains a sentence $X \rightarrow A$ (thus, X is taken to be the largest common subtype possible).

2.2 A proposed solution

Let us try to develop a learning algorithm for the Schema Inference Problem, learning from positive examples only. From the consistency criterion, we derive an obvious choice for a schema population Π based on the examples: for any type A in the domain, take $\Pi(A)=\{a \mid A(a)$ is a positive example$\}$. Add auxiliary types for non-empty intersections of type populations, that are not equal to existing type populations. The resulting set of populations is partially ordered by set-inclusion, and the diagram of this partial ordering represents a consistent schema[2]. As an illustration of this procedure,

[2]If two types in the domain have the same population, they are represented by distinct circles, connected by arrows in both directions.

suppose the domain of the schema is simply $\{A,B\}$. Initially, there are no examples, so the initial population Π_0 is defined by $\Pi_0(A)=\Pi_0(B)=\varnothing$. From this population we derive the following schema:

i.e., initially all types are considered equal. Now suppose the first example is $A(a)$: thus, $\Pi_1(A)=\{a\}$ and $\Pi_1(B)=\varnothing$, and we obtain the schema

After the example $B(b)$, the hypothesis is that both types have nothing in common:

Let the third example be $A(b)$. We obtain $\Pi_3(A)=\{a,b\}$ and $\Pi_3(B)=\{b\}$, and again we derive the hypothesis $\{B\rightarrow A\}$. If the fourth example is $B(b)$, we have $\Pi_4(A)=\Pi_4(B)=\{a,b\}$, and we are back at our initial hypothesis $\{B\rightarrow A, A\rightarrow B\}$. Adding a fifth example $B(c)$ gives $\Pi_5(A)=\{a,b\}$, $\Pi_5(B)=\{a,b,c\}$ and the schema

There are two points to be made here. First, without having defined anything like 'convergence' for our learning algorithm[3], it is intuitively clear that the procedure just illustrated does not 'smoothly' converge: it switches easily between the hypotheses '*B* is a subtype of *A*', '*B* and *A* are the same type', and '*A* is a subtype of *B*', without ever settling on one of these. Furthermore, at one time the algorithm proceeds from $\{A\rightarrow B\}$ to $\{A\rightarrow B, B\rightarrow A\}$, and at another time it proceeds in the opposite direction. This is counter-intuitive: adding more positive examples should lead to more (i.e. not-less) general hypotheses. To make this a little bit more precise, we define the notion of generality.

2.3 The generality ordering

The usual notion of generality is extensional, i.e. based on extensions (sets of instances) of expressions. In the case of schemas, instances are schema populations. So we define: given a type set σ, a schema Σ_1 over σ is *at least as general as* a schema Σ_2 over σ, notation $\Sigma_1\geq\Sigma_2$, iff each schema population[4] for Σ_2 is also a schema population for Σ_1. Alternatively, we say that Σ_2 is at least as specific as Σ_1, and write $\Sigma_2\leq\Sigma_1$. Obviously, this relation is reflexive and transitive; strictly speaking, it is not anti-symmetric: there are several schemas for schema populations that contain at least three identical type populations. Because this is a bit unlikely to occur in practice, we will treat $\geq$ as a partial ordering[5]. If $\Sigma_1\geq\Sigma_2$ but $\Sigma_1\neq\Sigma_2$, we write $\Sigma_1>\Sigma_2$ ($\Sigma_2<\Sigma_1$) and say that Σ_1 is *more general than* Σ_2 (Σ_2 is *more specific than* Σ_1).

---

[3]This will be done in section 4.1.

[4]For technical reasons, this definition of generality assumes schema populations consisting of **non-empty** type populations. This assumption will be made throughout the paper.

[5]Technically speaking, this amounts to assuming that there exists a normal form such that every equivalence class of the quasi ordering $\leq$ contains exactly one member in normal form.

There are five distinct schemas over $\{A,B\}$, and their generality ordering can be depicted as in fig. 2.

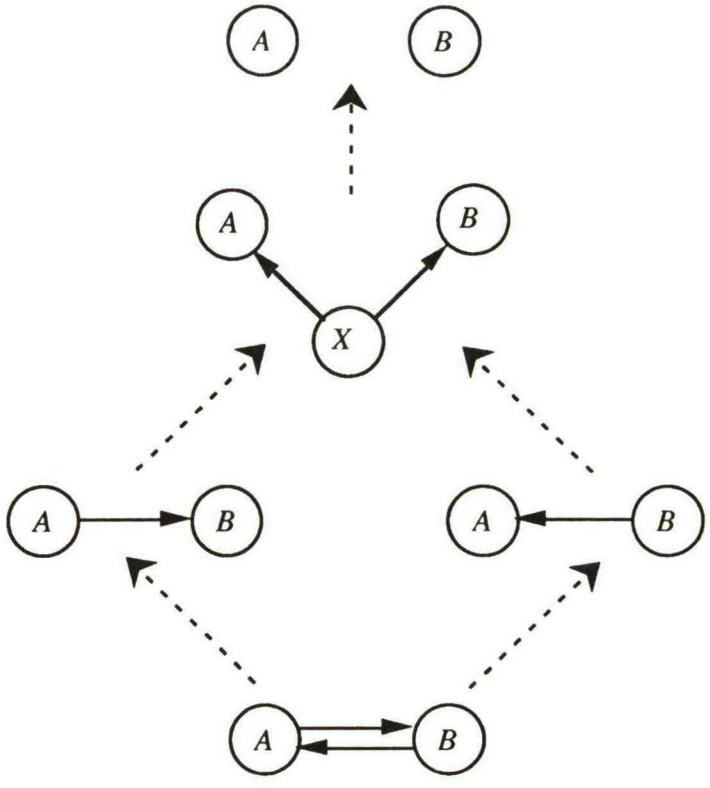


Figure 2. Schemas over $\{A,B\}$, partially ordered by generality.

The empty schema $\varnothing$ over $\{A,B\}$ is the most general schema, because any two type populations $\Pi(A)$ and $\Pi(B)$ constitute a schema population for it. A somewhat more specific schema is $\{X\rightarrow A, X\rightarrow B\}$; any population Π for this schema must satisfy $\Pi(A)\cap\Pi(B)\neq\varnothing$. More specific than this one but incomparable to each other are the schemas $\{B\rightarrow A\}$ and $\{A\rightarrow B\}$, with populations satisfying $\Pi(B)\subseteq\Pi(A)$ and $\Pi(A)\subseteq\Pi(B)$, respectively. Finally, the schema $\{B\rightarrow A, A\rightarrow B\}$ is the most specific schema, requiring $\Pi(A)=\Pi(B)$.

According to the above definitions, the population Π defined by $\Pi(A)=\{a,b\}$, $\Pi(B)=\{b\}$ is a population for the empty schema $\varnothing$ over $\{A,B\}$. However, intuitively we would expect that type populations for this schema would be disjoint, as the schema states that the two types have nothing in common. Conversely, given the schema population Π, we would intuitively say that it is typically a population for the schema $\{B\rightarrow A\}$. We can capture this intuition by defining a *typical (schema) population* Π_Σ for a schema Σ to be any schema population that is not a population for a more specific schema. That is, Π_Σ contains not more information than Σ conveys: adding information to Σ (making it more specific) causes Π_Σ to be no longer a population for it. If Π_Σ is a typical population for Σ, then we call Σ an *intended schema* for Π_Σ.

The naive learning algorithm above is precisely based on typical populations and intended schemas. The set of positive examples is interpreted as a typical population, for which an intended schema is construed. Notice, that any population has exactly one intended schema, a property about which we will have more to say later. The reason that the naive learning algorithm is somewhat unsatisfactory with respect to convergence, is that type populations in typical schema populations do not monotonically get

larger when the intended schema gets more general. For instance, let $\Pi_1(A)=\Pi_2(A)=\{a,b\}$ and $\Pi_1(B)=\{b,c\}$, $\Pi_2(B)=\{a,b,c\}$, thus $\Pi_2(B)\supset\Pi_1(B)$; yet, the intended schema for Π_1 is $\Sigma_1=\{X\rightarrow A, X\rightarrow B\}$, while the intended schema for Π_2 is $\Sigma_2=\{A\rightarrow B\}$, hence $\Sigma_2<\Sigma_1$. We will return to this issue as well.

The main conclusion must be, that what seems a reasonable and cautious approach at first sight, namely, to take only the positive and certain knowledge, and to translate it into a hypothesis by interpreting it as constituting a typical population, does not yield minimal hypotheses. On the other hand, in concept learning from examples it is exactly this approach of taking the disjunction of the positive examples, that does result in minimal hypotheses. Therefore, we proceed by studying in some detail the basic assumptions underlying the use of a partially ordered hypothesis space in concept learning. One of the most general accounts thereof is the Version Space model [Mitchell 1982]. After that, we analyse the differences between this model and our Schema Inference Problem, in order to come up with a model of second-order inductive learning.

3. Version Spaces

3.1 Outline of the Version Space model

Originally, the Version Space model (or VS model) was formulated as follows. Let there be given a set I of *instances* i, and a language L_G for expressing *generalisations* G. Generalisations describe sets of instances, and a generalisation G *matches* an instance i if i is a member of the set described by G, or more succinct, if i is an instance of G. Matching is described by a *matching predicate* $M(G, i)$, which is true iff i is an instance of G. Mitchell defines G_1 to be more specific than[6] G_2 iff $\{i \in I \mid M(G_1, i)\} \subseteq \{i \in I \mid M(G_2, i)\}$. That is, G_1 is more specific than G_2 (and, equivalently, G_2 is more general than G_1) iff the set of instances of G_1 is a subset of the set of instances of G_2.

Positive examples are instances of the generalisation to be learned, and *negative examples* are non-instances. Conversely, a generalisation is *consistent* with the examples iff it matches every positive example and no negative example. In determining the set of consistent generalisations, the generality ordering can be utilised as follows. If a generalisation G matches a positive example p, then every generalisation more general than G will also match p. Assuming that there are no infinite descending chains of generalisations, this implies that there exist minimal generalisations G_p consistent with p (i.e., a generalisation is consistent with p if and only if it is at least as general as some G_p). Similarly, under the assumption that every ascending chain of generalisations is finite, one can associate with each negative example n maximal generalisations G_n such that only and all generalisations at least as specific as some G_n are consistent with n.

From these considerations it follows that the set of generalisations consistent with every example, the *Version Space*, is bounded from below by a set S of most specific generalisations, derived from the positive examples, and bounded from above by a set G of most general generalisations, derived from the negative examples, such that a generalisation is consistent with the examples iff it is between S and G. This means, that the Version Space need not be stored explicitly: if the partial ordering is recursively enumerable, storage of S and G suffices. We further note that an efficient implementation of the VS model is possible, if there is an algorithm for computing new elements of S (G) out of G_p (G_n) and a new example e, when G_p (G_n) turns out to be inconsistent with e (without a mere search of the partial ordering).

3.2 Generalising the Version Space model

The Version Space model is a significant step towards a general theory of inductive learning. Many existing methods can be cast into the general framework of Version Spaces. However, the model presented above uses some rather specific notions, that are not essential for the model of Version Spaces. Also, there are some small technical shortcomings in Mitchell's formulation of the model. There is some confusion in the notion of partial order he uses: in fact, the generality ordering as defined above is a **quasi-ordering**, because several syntactically distinct generalisations may have the same set of instances. This raises a more general point: Mitchell fails to make a distinction between syntax and semantics. Indeed, what is presented to the learner are not instances, but **descriptions** of instances. Thus, Mitchell assumes that any instance can be uniquely described within the instance language, which

[6] In fact, the relation should be called *at least as specific as*, as has been pointed out before. However, here we follow Mitchell's account.

is not true in general. For a worked-out model which distinguishes between syntax and semantics, see [Laird 1988].

A more serious restriction of the VS model, as well as an opportunity to make the model more general, is indicated by Mitchell when he writes: "Notice the above definition of the [more-specific-than] relation is extensional—based upon the instance sets that the generalisations represent. In order for the more-specfic-than relation to be practically computable by a computer program, it must be possible to determine whether G_1 is more-specific-than G_2, without computing the (possibly infinite) sets of instances that they match." [Mitchell 1982, p.206]. In other words, there should be a syntactical ordering $\leq$ on generalisations, definable without reference to instances, with the **property** that $G_1 \leq G_2$ iff $\{i \in I \mid M(G_1, i)\} \subseteq \{i \in I \mid M(G_2, i)\}$. But then we could forget about instance sets altogether, and relate the matching predicate to the syntactical generality ordering as follows:

$$G_1 \leq G_2 \Leftrightarrow \forall i: M(G_1,i) \Rightarrow M(G_2,i) \tag{3.1}$$

This equivalence boils down to the following two implications:

$$\forall i: M(G_1,i) \wedge G_1 \leq G_2 \Rightarrow M(G_2,i) \tag{3.1a}$$

$$\neg(G_1 \leq G_2) \Rightarrow \exists i: M(G_1,i) \wedge \neg M(G_2,i) \tag{3.1b}$$

Formula (1a) expresses what is called completeness of the matching predicate (or consistency predicate) in [Flach 1989]: it enables us to say that any generalisation between the boundaries S and G is indeed consistent. Formula (1b) requires, for any two generalisations G_1 and G_2 such that not $G_1 \leq G_2$, the existence of a witness i that is matched by G_1 and not matched by G_2. In words: the matching predicate should not be too coarse for the partial ordering at hand. This implication can be rephrased into a formula describing the relation between syntax and semantics (see [Laird 1988]).

Formula (1a) can be generalised in several ways. First, notice that it can also be written as

$$\forall i: \neg M(G_2,i) \wedge G_1 \leq G_2 \Rightarrow \neg M(G_1,i) \tag{3.1a'}$$

stating that if G_2 does not match i, anything below it won't either. Although formulas (1a) and (1a') are logically equivalent, we could say that (1a) describes the existence of the lower boundary S, and (1a') describes the existence of the upper boundary G. To make this more apparent, the formulas can be written in the following form:

$$\forall i: M_S(G_1,i) \wedge G_1 \leq G_2 \Rightarrow M_S(G_2,i) \tag{3.2a}$$

$$\forall i: M_G(G_1,i) \wedge G_1 \geq G_2 \Rightarrow M_G(G_2,i) \tag{3.2b}$$

where M_S denotes the original matching predicate M, and M_G is defined as the negation of M_S[7]. Now, it has been shown in [Flach 1989] that there are other choices for M_G possible (that is, other than the **negation** of M_S), while retaining the VS model.

Secondly, the association of a **lower** boundary with positive examples and an **upper** boundary with negative examples is, in a certain sense, arbitrary. We call a model a *Generalised Version Space model* or GVS model if the space of consistent hypotheses (the *Generalised Version Space* or VS_g) is bounded from above and below in any way, such that every generalisation between these boundaries is consistent with the examples. In a Generalised Version Space model, positive examples could result in an upper boundary; alternatively, it might be the case that a boundary can only be associated with both

[7]Thus, the VS model satisfies the separability condition (see section 4.3).

positive and negative examples. Clearly, the VS model is a Generalised Version Space model, with a lower boundary according to positive examples and an upper boundary according to negative examples.

It is the generalisation to Generalised Version Space models that applies in the case of the Schema Inference Problem. This will be detailed in the next sections, in which we will try to capture the essential characteristics of the Schema Inference Problem in a formal model.

4. Second-order inductive learning

4.1 Formal definitions

It has been shown in the previous section, that the idea of instances of generalisations playing the role of examples underlies the development of the Version Space model. Although it can be generalised in several ways, it certainly plays a crucial role in concept learning from examples. It has also been shown, that the VS model can not be applied to the Schema Inference Problem. What, then, are the intuitions behind the Schema Inference Problem?

In the Schema Inference Problem, schemas are the generalisations. Instances of schemas are schema populations. Examples are elements of type populations, thus elements of schema populations. Pictorially, we have the following situation:

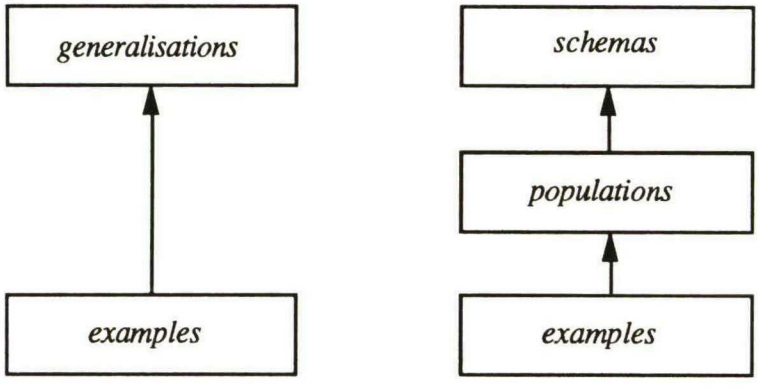


Figure 3. Layers occurring in *a.* concept learning from examples and *b.* the Schema Inference Problem

In the Schema Inference Problem, there is an extra layer between the examples and the generalisations, and therefore we call the kind of learning that occurs in solutions to the Schema Inference Problem *second-order inductive learning*. In this context, traditional concept learning from examples would be called *first-order* inductive learning (and rote learning might be called zeroth-order inductive learning).

Let us make the nature of second-order inductive learning more precise. In doing so, we will depart slightly from Mitchell's terminology by preferring the term 'hypothesis' over 'generalisation'. Let there be given a set $\boldsymbol{H}$ of *(second-order) hypotheses* H; with each hypothesis H, a set $[H]$ is associated, called the *extension* of H. Elements of $[H]$ are called *populations* for H (in first-order learning, elements of $[H]$ would be called instances of H). We will assume, that there is a generality (quasi-)ordering $\leq$ on $\boldsymbol{H}$, and employ the usual notation and terminology. Also, we assume that this quasi-ordering corresponds to the partial ordering of extensions, i.e. $H_1 \leq H_2$ iff $[H_1] \subseteq [H_2]$. Additionally, if $[H_1] \subset [H_2]$ we write $H_1 < H_2$ and say that H_1 is more specific than H_2 (H_2 is more general than H_1). If $[H_1]=[H_2]$, we call H_1 and H_2 *variants*. As a result of these assumptions, any population for a hypothesis H is also a population for any hypothesis more general than H.

Thus far, the only difference with first-order learning is terminology. We now assume that each population for a hypothesis is itself a set; its elements are called *instances*, and the set of instances is denoted I_H. Thus: hypotheses denote sets of populations, and populations are sets of instances. An *example* is an element of $I_H \times \{+,-\}$; a pair $\langle p,+\rangle$ is called a *positive* example, a pair $\langle n,-\rangle$ is called a *negative* example (where p and n denote the instances involved in the examples). A *second-order inductive learning task* $\langle \boldsymbol{H}, \boldsymbol{E}\rangle$ is characterised by a hypothesis space $\boldsymbol{H}$ (where extensions and instances are implicitly understood) and a set of examples $\boldsymbol{E}$. The consistency conditions for a second-order inductive learning task $\langle \boldsymbol{H}, \boldsymbol{E}\rangle$ can now be stated as follows: a population P for a hypothesis $H \in \boldsymbol{H}$ is *consistent with an example* e iff either $e=\langle p,+\rangle$ and $p \in P$ or $e=\langle n,-\rangle$ and $n \notin P$; P is *consistent with a set of examples* $\boldsymbol{E}$ iff P is consistent with each example $e \in \boldsymbol{E}$. A hypothesis $H \in \boldsymbol{H}$ is *consistent with a set of examples* $\boldsymbol{E}$ iff there is a population for H that is consistent with $\boldsymbol{E}$[8].

Note carefully, that this definition of consistency of hypotheses requires the existence of a **single** population that is consistent with **every** example. That is, H is consistent with e_1 if there is a population $P_1 \in [H]$ such that P_1 is consistent with e_1; similarly, H may be consistent with e_2 by virtue of another population $P_2 \in [H]$, consistent with e_2. Still, this does **not** entail that H is consistent with $\{e_1,e_2\}$, because it is conceivable that $[H]$ contains no single population consistent with both e_1 and e_2. We call the property of a hypothesis H being consistent with a set of examples $\boldsymbol{E}$ iff H is consistent with each example in E, the *property of compositionality*. It is an important property, because it allows for incremental learning algorithms that need not reconsider all previous examples, once an inconsistency is detected. While in first-order inductive learning the property of compositionality holds by definition, it need not be valid *a priori* in every second-order learning problem. For instance, it is not valid for the Schema Inference Problem.

For judging the *correctness* of a learning algorithm, i.e. its capability to infer the correct hypothesis eventually, consistency is not enough. As an example, in first-order learning the strategy to take a most general consistent hypothesis is incorrect when only positive examples are available, because in that case it will stick to the most general hypothesis forever, and thus will fail to come up with the correct hypothesis (unless it is the most general one). On the other hand, a strategy to take a most specific consistent hypothesis is correct in this case, provided that such a hypothesis is unique once enough examples are available. Several models for correctness of inductive algorithms have been proposed. One of the best-known models is identification in the limit [Gold 1967, Angluin & Smith 1983]. An inductive algorithm *identifies* the correct hypothesis *in the limit*, iff it makes the correct guess after a finite amount of time, and never changes its guess afterwards. To this end, the algorithm is supplied with a *sufficient presentation*, i.e. a sequence of examples such that every instance occurs at least once. The algorithm is not required to signal its final guess (if it does, it *finitely identifies* the correct hypothesis); hence, for all practical applications restrictions are applied to the global convergence of the sequence of hypotheses. A common restriction is *consistency*, i.e. any hypothesis should be consistent with the examples seen so far. This restriction leads to algorithms of which the intermediate hypotheses make sense. Another common restriction is, that the algorithm be *conservative*, that is, it outputs a hypothesis different from its previous guess only when the previous guess is inconsistent with the examples seen so far.

How do these criteria apply to the naive learning algorithm (henceforth referred to as the *NLA*) for the Schema Inference Problem? The first question is, whether the algorithm is correct. Let Σ_0 be the correct schema. In order to give examples for this schema, the teacher selects a population Π_0 for Σ_0. If the teacher supplies a sufficient presentation, eventually every pair (A,a), where $a \in \Pi_0(A)$, will have been presented as an example $A(a)$. But then the NLA has identified Π_0, because it constructs the minimal

[8]For brevity, if the set of examples $\boldsymbol{E}$ is understood we talk about consistent populations and hypotheses.

population from the examples. For this population, the NLA constructs the intended schema. This schema will only be equal to Σ_0, if Π_0 is a typical population for Σ_0. Hence, we can draw the conclusion that the NLA is correct iff the teacher selects examples according to a typical population for the correct hypothesis. This constraint seems fairly reasonable.

The NLA is also consistent: at any stage, the current hypothesis Σ has a population (namely, its typical population Π_Σ) such that for every example $A(a)$, $a \in \Pi_\Sigma(A)$. However, the algorithm is not conservative, as can easily be concluded from the illustration given in section 2.2, where the hypothesis $\{B \rightarrow A\}$ is first adopted, then abandoned, only to be adopted again later. Because the hypothesis is adopted a second time and the algorithm is consistent, there is a population corresponding to all the examples given until then; but then the hypothesis is also consistent with any subset of the examples given, and it follows that there was no need to abandon it in the first place. Because the NLA is not conservative, it does not exhibit a 'smooth' convergence towards the correct schema.

4.2 Second-order learning of hypotheses by first-order learning of prototypes

As may have become apparent in the previous section, the naive learning algorithm presented in section 2.2 embodies a particular implementation technique for second-order inductive learning. As has been shown, the algorithm contains a correct (as well as consistent and conservative) procedure for first-order learning of populations. In a second stage, the inferred population is mapped to a uniquely determined (because intended) schema. Obviously, this mapping is only justified if the original population is a typical population for the correct schema. The underlying assumptions can be generalised as follows.

> PROTOTYPE ASSUMPTION. *Some populations have unique minimal (with respect to the generality ordering $\leq$) hypotheses for which they are populations. Such populations are called* prototypes, *and the corresponding minimal hypothesis for a prototype is called its* intended hypothesis. *There exists an effective procedure for calculating the intended hypothesis for a given prototype.* ■

The idea of the Prototype Assumption is, that if the teacher uses a prototype for selecting examples, the learner can learn by first-order identification of the prototype from the examples, followed by the determination of the intended hypothesis for that prototype. *First-order identifiability* (of populations) refers to the existence of methods for identification in the limit of any population from positive and negative examples (involving instances). Similarly, *second-order identifiability* (of hypotheses) refers to the existence of methods for identification in the limit of any hypothesis from positive and negative examples (involving instances). We thus arrive at the following proposition.

> PROPOSITION 1. *Under the Prototype Assumption, first-order identifiability of prototypes implies second-order identifiability of intended hypotheses.* ■

Interestingly, under the Prototype Assumption it is possible to identify a hypothesis in the limit without being able to identify a single prototype in the limit. Define the relation $\equiv$: $P \equiv P'$ iff any two prototypes P and P' have the same intended hypothesis. Clearly, this relation is an equivalence relation, and each of its equivalence set contains all prototypes for a specific hypothesis. Suppose it is possible to identify some hypothesis H_0 from examples, then it may still be the case that no first-order inductive algorithm is able to distinguish some elements of the set of prototypes for H_0.

PROPOSITION 2. *Under the Prototype Assumption, second-order identifiability is at least as strong as first-order identifiability.* ■

It should be obvious by now that Proposition 1 describes the approach exemplified by the naive learning algorithm: in the Schema Inference Problem, every population is a prototype (a typical population in the terminology of secion 2.3) for its intended schema. Notice also, that in the Schema Inference Problem there are indeed several prototypes for every hypothesis. As we have seen, this causes the second-order learning of hypotheses by means of first-order learning of prototypes to be non-conservative, even if the first-order learning is conservative. Another drawback of this approach is, that no advantage can be taken of the generality ordering $\leq$ on the hypothesis space, if this ordering does not correspond to the partial ordering by set inclusion of populations, as we have seen in the Schema Inference Problem. In the next section, we study a method for implementing second-order inductive learning directly.

4.3 Second-order learning in a Generalised Version Space model

The notion of a Generalised Version Space model has already been introduced. The idea is, that positive examples do not necessarily result in most specific hypotheses and thus a lower boundary of the Version Space; nor do negative examples necessarily result in an upper boundary. Any other set of boundaries could be equally useful as the VS model. In this section, it will be shown that second-order inductive learning satisfies a GVS model without satisfying the VS model. This requires the following:

(*i*) there exist minimal/maximal consistent hypotheses such that no hypothesis below/above one of these is consistent with the examples;

(*ii*) every hypothesis between one of the minimal consistent hypotheses and one of the maximal consistent hypotheses is consistent with the examples.

As has been remarked earlier, Mitchell's Version Spaces are VS_g's. In addition, they satisfy the *separability condition*: consistency can be split into *upper* consistency and *lower* consistency, such that any hypothesis is minimal consistent iff it is minimal lower consistent and upper consistent, and any hypothesis is maximal consistent iff it is maximal upper consistent and lower consistent. In the VS model, lower consistency means consistency with positive examples, and upper consistency means consistency with negative examples. If a Generalised Version Space model satisfies the separability condition, condition (*ii*) above can be split into two parts:

(*ii*a) every hypothesis above one of the minimal consistent hypotheses is lower consistent with the examples;

(*ii*b) every hypothesis below one of the maximal consistent hypotheses is upper consistent with the examples.

The following result shows, that second-order inductive learning satisfies a Generalised Version Space model.

THEOREM 3. *Assuming that every ascending or descending chain in the hypothesis space is finite, second-order inductive learning satisfies a Generalised Version Space model.*

Proof. The following has to be proven: if $H_{\min}$ is a minimal consistent hypothesis and $H_{\max}$ is a maximal consistent hypothesis and $H_{\min} \leq H \leq H_{\max}$, then H is a consistent hypothesis. Assuming the existence of minimal and maximal consistent hypotheses, this is logically equivalent with

$$H_1 \text{ is consistent} \wedge H_2 \text{ is consistent} \wedge H_1 \leq H \leq H_2 \Rightarrow H \text{ is consistent}$$

According to the definitions, a hypothesis H is consistent iff there is a population P for H that is consistent. But then P is also a population for any $H' \geq H$, hence any $H' \geq H$ is also consistent:

$$H_1 \text{ is consistent} \wedge H_1 \leq H \Rightarrow H \text{ is consistent}$$

Obviously this latter formula implies the former. ■

The latter formula also implies that the VS_g is only bounded from above by the most general hypotheses, and that this upper boundary is **fixed**. This means, that convergence has to be provided by the lower boundary moving upwards alone. In other words, second-order inductive learning trivially satisfies the separability condition: in practice, we only work with the lower boundary.

Why are boundaries of a Version Space useful? The appropriate answer, of course, is that thase boundaries move toward each other as learning proceeds, excluding more and more hypotheses. Indeed, would this be not true for a particular learning problem, then we would have severe doubts concerning the well-definedness of the problem. We therefore define a problem of inductive learning to be *sound* iff any hypothesis that becomes inconsistent after a number of examples, remains inconsistent when new examples are added. The following result shows that second-order inductive learning is sound. In stating this Theorem, we use the notions of *positive instance set* $PE=\{p \in \boldsymbol{I_H} \mid \langle p,+\rangle \text{ is a positive example}\}$ and *negative instance set* $NE=\{n \in \boldsymbol{I_H} \mid \langle n,-\rangle \text{ is a negative example}\}$.

THEOREM 4 (Soundness of second-order inductive learning). *Any hypothesis that is inconsistent with positive instance set PE or negative instance set NE, will also be inconsistent with any larger positive instance set* $PE' \supset PE$ *resp. any larger negative instance set* $NE' \supset NE$.

Proof. H is inconsistent iff for every population P for H, $PE \not\subseteq P \vee P \not\subseteq (\boldsymbol{I_H} - NE)$, which implies for any $PE' \supseteq PE$ and any $NE' \supseteq NE$, $PE' \not\subseteq P \vee P \not\subseteq (\boldsymbol{I_H} - NE')$ for every population P for H. ■

Finally, we investigate the condition under which a hypothesis $H_{\min}$ is a minimal consistent hypothesis. Let $P_{\min}$ be a population for $H_{\min}$, and let H be a hypothesis below $H_{\min}$, then any population P for H should be inconsistent:

$$PE \subseteq P_{\min} \subseteq (\boldsymbol{I_H} - NE) \wedge H < H_{\min} \Rightarrow PE \not\subseteq P \vee P \not\subseteq (\boldsymbol{I_H} - NE) \qquad (4.1)$$

Until more is known about the relation between populations and hypotheses, nothing more can be said. If, for instance, smaller hypotheses would have smaller populations, the lower boundary would be associated with positive examples (as in first-order learning), and any hypothesis with minimal populations P such that $PE \subseteq P$ would belong to this boundary. Similarly, if smaller hypotheses have larger populations, the lower boundary would be associated with negative examples. Of course, a lower boundary could exist even if the separability condition does not apply.

4.4 Second-order inductive learning for the Schema Inference Problem

In this section, we will develop a learning algorithm for the Schema Inference Problem, based on the GVS approach. This learning algorithm will be conservative, as opposed to the naive learning algorithm given in section 2.2, and thus converge more smoothly to the final solution. As suggested in the previous section, we have to establish the relation between populations and schemas, in order to construct a minimal consistent schema. Recall that a schema is consistent if it has a population containing every positive example and no negative example. It follows thaat a **minimal** consistent schema must have a **typical** population containing every positive example and no negative example (otherwise, there would be a more specific schema for this population, which would also be consistent). In the naive learning algorithm, we tried to find this minimal consistent schema by constructing a **smallest** population agreeing with the examples (i.e., the positive instance set PE[9]). However, this rests upon the assumption that smaller populations have more specific intended schemas, which is not true in general. For instance, $\Pi_1=\{A(a),A(b),B(b),B(c)\}$ is a prototype for $\Sigma_1=\{X\rightarrow A, X\rightarrow B\}$, and $\Pi_2=\{A(a),A(b),B(a),B(b),B(c)\}$ is a prototype for $\Sigma_2=\{A\rightarrow B\}$: $\Pi_1\subset\Pi_2$, and $\Sigma_1>\Sigma_2$. This can be formalised as follows.

THEOREM 5. *Let Π be a population containing the type symbol A and the constant symbol b, and let Σ be its intended schema. If Σ' is the intended schema for $\Pi\cup\{A(b)\}$, then $\Sigma'\leq\Sigma$.*

Proof. If $A(b)\in\Pi$, then $\Sigma'=\Sigma$. If $A(b)\notin\Pi$, then there is a type symbol B such that $B(b)\in\Pi$. Thus, adding $A(b)$ to Π increases the number of individuals A and B have in common. Withouth loss of generality, we may assume that Σ is over $\{A,B\}$ (see fig. 2). We can distinguish the following cases:

(i)	$\Sigma=\emptyset$;	(a) $\Sigma'=\{X\rightarrow A, X\rightarrow B\}$	(b) $\Sigma'=\{B\rightarrow A\}$
(ii)	$\Sigma=\{X\rightarrow A, X\rightarrow B\}$;	(a) $\Sigma'=\{X\rightarrow A, X\rightarrow B\}$	(b) $\Sigma'=\{B\rightarrow A\}$
(iii)	$\Sigma=\{A\rightarrow B\}$;	(a) $\Sigma'=\{A\rightarrow B\}$	(b) $\Sigma'=\{A\rightarrow B, B\rightarrow A\}$
(iv)	$\Sigma=\{A\rightarrow B, B\rightarrow A\}$ and $\Sigma'=\{A\rightarrow B, B\rightarrow A\}$.		

■

The condition that the constant symbol b is already contained in Π is crucial, because otherwise the resulting schema might indeed be more general. E.g., if $\Pi=\{A(a),B(a)\}$, then $\Sigma=\{A\rightarrow B, B\rightarrow A\}$, but $\Pi\cup\{A(b)\}=\{A(a),B(a),A(b)\}$, hence $\Sigma'=\{A\rightarrow B\}$ and $\Sigma'>\Sigma$. Theorem 5 can be paraphrased as: **larger prototypes**[10] **have more specific intended schemas.**

COROLLARY 6. *Every schema more specific than a given schema Σ can be obtained by augmenting a prototype Π for Σ with typed individuals, of which both type and individual occur in Π.*

Proof. See cases *(i)*-*(iv)* of Theorem 5. ■

Due to the fact that compositionality does not hold for the Schema Inference Problem, a second-order learning algorithm for it can not be incremental. At each stage, we have to use all previous examples to build a consistent schema. Corollary 6 suggests the following method for obtaining a minimal consistent schema: augment the positive instance set PE to a population Π that assigns every individual in PE to every type in the type set, and construct an intended schema for Π–NE. For instance,

[9] In this section, we specify populations for a schema by sets like $\{A(a),B(a),\ldots\}$, in order to keep on using set-inclusion among populations.

[10] That is, if we compare prototypes that contain exactly the same type and constant symbols.

if the examples are $A(a)$, $B(b)$ and $\neg A(b)$, then we have $\Pi=\{A(a),A(b),B(a),B(b)\}$ and $\Pi-NE=\{A(a),B(a),B(b)\}$, such that $\{A\rightarrow B\}$ is the minimal consistent schema. There is however one caveat: if there are many negative examples, then some types may have no individuals in common in Π–NE. But it is always possible that such an individual will be introduced in a new positive example. Therefore, in the final prototype Π_Σ we include a typed individual $A(x)$ for every type symbol A in the type set (where x is a reserved individual symbol, not present in the examples). E.g., if the examples are $A(a)$, $B(b)$, $\neg A(b)$, $\neg B(a)$, then we build the 'maximal' consistent prototype $\{A(a),A(x),B(b),B(x)\}$, with intended schema $\{X\rightarrow A, X\rightarrow B\}$. Notice that this is indeed a minimal consistent schema, as opposed to $\varnothing$, which we would have obtained had we not included $A(x)$ and $B(x)$ in our prototype. Notice also, that there is obviously no way of arriving at $\varnothing$ as a minimal consistent schema, if the number of constant symbols is uncountable.

Note carefully, that we have not yet made any assumption about whether the teacher chooses his examples from a typical population or just from any population for the target schema. The fact that the most specific consistent schemas are consistent by virtue of prototypes is just a consequence of the model itself. Therefore, the assumption that the teacher chooses his examples according to a typical population does not make any difference for the lower boundary. It does, however, make some difference for the upper boundary: once it has been established that in a **prototype** two types have a common member, a maximal consistent hypothesis should at least state that these two types have a common subtype. However, this is a hypotheses that can never be falsified, ands so it will remain the maximal consistent hypothesis forever. Convergence has not improved much.

Let us state the convergence properties for second-order learning for the Schema Inference Problem more clearly. In general, the VS_g will not collapse to a unique solution, because in many cases the boundaries never meet. So we have two learning strategies: stick to the lower boundary, and stick to the upper boundary. If we stick to the lower boundary, the resulting algorithm performs identification in the limit provided the teacher selects examples according to a prototype, and provided the correct schema is **connected**. Moreover, the resulting algorithm is consistent and conservative, thus providing 'smoother' convergence than the NLA. If we stick to the upper boundary, the resulting algorithm performs identification in the limit provided the teacher selects examples according to a prototype, and provided the types in the correct schema are either unconnected or only connected via common subtypes (two rather uninteresting cases).

Finally, we illustrate the second-order inductive learning algorithm for the Schema Inference Problem roughly sketched in this section. Let the type set be $\{A,B\}$. In fig. 4 below, each line specifies the example given, the maximal consistent hypothesis after that example, the minimal consistent hypothesis after that example, and the prototype used to construe that minimal consistent hypothesis.

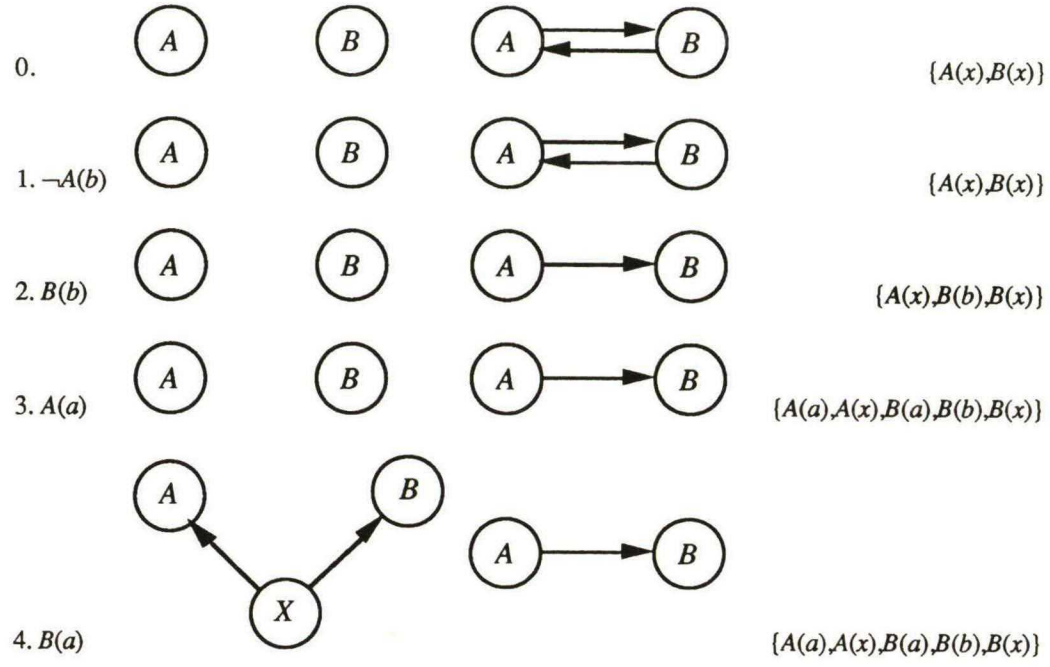


Figure 4. Learning process for the Schema Inference Problem using the second-order inductive learning algorithm.

Initially, VS_g is equal to the entire space of hypotheses. The first, negative example has no effect on VS_g. The second, positive example concerns the same constant symbol as the first, negative example, and causes the lower boundary to move upwards. The third, positive example again has no effect, but in combination with the fourth, positive example it causes the upper boundary to move downwards. Should we add two more examples, $A(c)$ and $\neg B(c)$, then the upper and lower boundary would collapse, and the single solution would be $\{X \rightarrow A, X \rightarrow B\}$. Notice, that the upper boundary moves downward due to positive examples, and the lower boundary moves upward due to the combination of positive and negative examples.

5. Second-order learning and logic

Until now, we have contrasted second-order inductive learning with the standard framework of Version Spaces, and we have concluded that there are many differences. This is an important result, because it allows us to incorporate the Version Space model in a more general 'meta-model'. On the other hand, a formulation of the Schema Inference Problem in first-order logic[11] does not seem (at first sight) to cause problems. An example is a ground literal like `p(a)` (positive example) or ¬`q(b)` (negative example); a schema consists of formulas of the form `p(X) :-q(X)`, and a schema S is consistent with a set of examples E iff $S,E \not\models \square$. In this section, we briefly investigate whether existing methods for induction in first-order logic are applicable to the Schema Inference Problem.

A general framework for inductively inferring logical theories from facts was provided by [Shapiro 1981]. In this framework, the induction algorithm starts with the most general theory $\square$ (which implies everything), and a new example is read. If the current theory is too strong (implies too much), then the guilty clause is diagnosed and removed from the theory. If the current theory is too weak, then a new clause has to be added to the theory; candidates are socalled *refinements* of previously removed clauses. In order to guarantee identification in the limit of the theory, the refinement operator must be *complete*. A complete refinement operator for the Schema Inference Problem is shown in fig. 4.

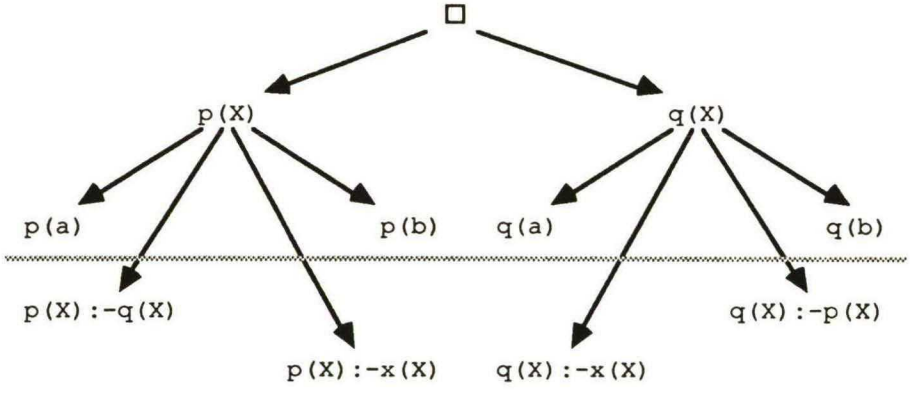


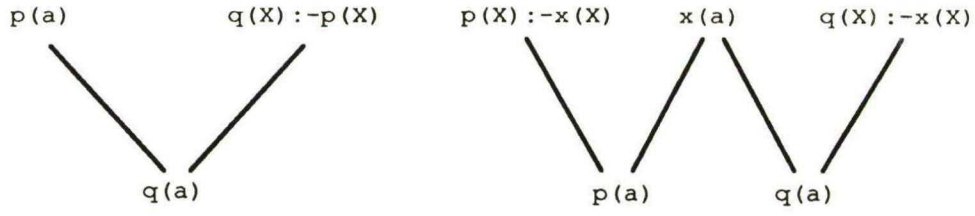
Figure 4. A complete refinement operator for the Schema Inference Problem.

Despite appearances, there is a problem here: the clauses below the dashed line in fig. 4 will never be induced, because the refinements that yield a smaller increase in the size of the current theory are tried first. In other words, the induction algorithm will do nothing more than collecting the examples. The reason, of course, is the definition of consistency: in Shapiro's framework, a theory is consistent with a set of facts if it **implies** these facts, while in our framework, we use the weaker property of **logical** consistency.

An alternative, but equally general (although somewhat less elaborated formally) framework for induction of logical theories is presented in [Muggleton & Buntine 1988]. In this framework, induction is carried out by inverting resolution. A number of inverse resolution operators is defined, including the *V*-

[11]The term 'second-order' as applied to inductive learning should not be interpreted in the logical sense of 'second-order predicates', although these issues are related: in the Schema Inference Problem, we are trying to find out if the second-order predicate 'subtype of' is satisfied.

operator, which induces `q(X):-p(X)` from `p(a)` and `q(a)`, and the *W-operator*, which induces `p(X):-x(X)`, `x(a)`, and `p(X):-x(X)` from `p(a)` and `q(a)` (fig. 5).



(a) V-operator (b) W-operator

Figure 5. Inverse resolution operators for the Schema Inference Problem.

Of course, there is a control regime that decides the order in which the inverse resolution operators are tried in a given situation. Because of our non-standard definition of consistency, we can not use Muggleton and Buntine's control regime directly. On the other hand, it seems very well possible to incorporate the idea of inverse resolution operators in the framework of second-order inductive learning.

6. Concluding remarks

In this paper, we have presented a new paradigm for inductive learning. The usefulness of this paradigm was suggested by the Schema Inference Problem, which we used to define the paradigm. We have sketched methods for devising learning algorithms for second-order inductive learning, one based on traditional first order learning, and a new method specifically for second-order learning.

In the course of the paper, we have pointed at several possibilities for generalising the Version Space model. One of these generalisations we called Generalised Version Spaces or VS_g's, in which upper and lower boundaries may differ from the Version Space boundaries. We have also identified the separability condition, which allows consistency to be separated into upper and lower consistency, and the soundness property of inductive learning. Another important notion is compositionality, which allows for incremental learning algorithms. We have shown, that compositionality does not hold (in general) in second-order inductive learning. In our ongoing research [Flach 1990a], we are merging these notions into a general model.

We have shown, that second-order inductive learning not only differs from the Version Space model, but also from the framework of induction of logical theories from facts, with respect to the consistency criterion used. Consequently, Shapiro's methods are not applicable (at least not without modification), while Muggleton and Buntine's inverse resolution operators are, by deriving a new control regime under which they should be applied.

A final observation: an alternative account of second-order inductive learning can be given, if a population is viewed as an example, that is only incompletely specified by the instances. In [Flach 1989] we argued that concept learners should allow for incomplete examples, because they still provide some information, albeit less than a completely specified example. From this perspective, second-order inductive learning as defined here could be called concept learning from one incomplete example. This account of second-order inductive learning possibly makes an integration of first-order and second-order learning into a general theory of inductive learning more feasible.

References

[Angluin & Smith 1983] D. ANGLUIN & C.H. SMITH, 'Inductive inference: theory and methods', *Computing Surveys* 15:3, 238-269.

[Flach 1989] P.A. FLACH, 'On the significance of examples in inductive learning', unpublished manuscript.

[Flach 1990a] P.A. FLACH, 'Towards a meta-theory of inductive learning', ITK Research Report, Institute for Language Technology & Artificial Intelligence, Tilburg University, the Netherlands (forthcoming).

[Flach 1990b] P.A. FLACH, 'Inductive methods in data modeling', ITK Research Report, Institute for Language Technology & Artificial Intelligence, Tilburg University, the Netherlands (forthcoming).

[Gold 1967] E.M. GOLD, 'Language identification in the limit', *Information and Control* 10, 447-474.

[Laird 1988] P.D.LAIRD, *Learning from good and bad data*, Kluwer, Boston.

[Mitchell 1982] T.M. MITCHELL, 'Generalization as search', *Artificial Intelligence* 18, 203-226.

[Muggleton & Buntine 1988] S. MUGGLETON & W. BUNTINE, 'Machine invention of first-order predicates by inverting resolution', *Proc. Fifth Int. Conf. on Machine Learning*, Morgan Kaufmann, San Mateo.

[Shapiro 1981] E.Y. SHAPIRO, *Inductive inference of theories from facts*, Techn. rep. 192, Comp. Sc. Dep., Yale University.

[Vermeir & Nijssen 1982] D. VERMEIR & G.M. NIJSSEN, 'A procedure to define the object type structure of a conceptual schema', *Information Systems* 7:4, 329-336.

ITK: P.O. BOX 90153 5000 LE TILBURG THE NETHERLANDS