The Knowledge Engineering Review, Vol 7:3, 1992, 251–269

A unifying framework for concept-learning algorithms

LUC DE RAEDT and MAURICE BRUYNOOGHE

Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium

Abstract

A unifying framework for concept-learning, derived from Mitchell's Generalization as Search-paradigm, is presented. Central to the framework is the generic algorithm Gencol. Gencol forms a synthesis of existing concept-learning algorithms as it identifies the key issues in concept-learning: the representation of concepts and examples, the search strategy and heuristics, and the operators that transform one concept-description into another one when searching the concept description space. Gencol is relevant for practical purposes as it offers a solid basis for the design and implementation of concept-learning algorithms. The presented framework is quite general as seemingly disparate algorithms such as TDIDT, AQ, MIS and version spaces fit into Gencol.

1 Introduction

Concept-learning from positive and negative examples occupies a central position in the field of machine learning. Roughly speaking, concept-learning techniques aim at finding a definition of an unknown concept from positive and negative examples of the concept. The definition should allow us to distinguish positive from negative examples. Concept-learning techniques have been successfully applied in a wide range of situations such as learning heuristics for symbolic integration (Mitchel *et al.*, 1983), deriving diagnostic rules for soy-bean diseases (Michalski *et al.*, 1985), synthesizing new knowledge from examples (Bratko *et al.*, 1989), etc. Learning concepts from examples has developed into a broad area of research; numerous concept-learning techniques are described in the literature see, e.g., (Michalski *et al.*, 1983, 1986; Kodratoff & Michalski, 1990). As for other broad research areas, distinguished researchers have analysed different algorithms in order to extract the underlying principles, to point out the similarities and differences between various approaches and to explain systems independent of their implementation or application (Mitchell, 1982; Michalski, 1983; Dietterich & Michalski, 1983; Laird, 1986; Angluin & Smith, 1983; Gams & Lavrač, 1987; Shapiro, 1981; Bundy *et al*, 1985; Rendell, 1986; Kalkanis & Conroy, 1991). Although this work led to a better understanding of the field, most of it focussed on analysis rather than synthesis. Indeed, concept-learning has been analysed as a search-problem (Mitchell, 1982) and as an application of generalization and specialization rules (Mitchell, 1982; Laird, 1986; Michalski, 1983). Furthermore, a few comparative surveys have been made (Bundy *et al.*, 1985; Dietterich & Michalski, 1985; Cohen & Feigenbaum, 1981; Langley & Carbonell, 1987; Kalkanis & Conroy, 1991).

As opposed to these analytical investigations, we present a synthesis of concept-learning in the form of the generic concept-learning algorithm Gencol (GENeric COncept-Learning algorithm). By the term *generic* we mean that the important properties of specific algorithms appear as (generic) parameters in Gencol. Instantiating these parameters with particular values defines a particular algorithm. Gencol builds on Mitchell's (1982) landmark paper, *Generalization as search*, as Gencol makes abstraction of the search-strategy and the description languages. We conjecture that Gencol can be instantiated to most existing concept-learning algorithms using generalization and specialization (this excludes genetic learning and simulated annealing). To support this claim

we have shown (De Raedt, 1991) how it can be instantiated to obtain seemingly disparate systems: the TDIDT-family (Quinlan, 1986), the Model Inference System (Shapiro, 1983), the AQ methodology (Michalski, 1983) and the candidate elimination algorithm or version space approach of Mitchell (1977). The generic algorithm Gencol not only serves synthetic purposes but also analytic ones, since formulating different concept-learning algorithms in the same framework facilitates the understanding and appreciating of the similarities and differences between the various approaches. Furthermore, because Gencol conceptually synthesizes many concept-learning systems, it provides good introduction to the area of concept-learning and mirrors its principles.

Until now there has been little synthetical work on concept-learning, with a few notable exceptions (Shapiro, 1981; Gams & Lavrač, 1987; Michalski, 1983; Laird, 1986). The approach taken in this paper is unique in the sense that it is:

1 *general enough to cover (to the best of our knowledge) all algorithms that use specialization and generalization to search the concept description space, and*
2 *that it shows how seemingly disparate techniques are instantiations of the same algorithm.*

Gencol is also relevant to practitioners of artificial intelligence and knowledge engineering, as it provides a methodology for the development of concept-learning algorithms and applications. To design concept-learning algorithms, one only needs to make appropriate choices for Gencol's generic parameters. Moreover, within the presented framework, it is easy to experiment with several variants of a particular algorithm as it is possible to replace instantiations of generic parameters by other ones.

This paper is organized as follows: in section 2, the problem of concept-learning is analysed and defined; in section 3, the generic concept-learning algorithm Gencol is presented and explained; in the next section it is shown how Gencol can be instantiated to obtain a wide range of algorithms; in section 5 we briefly discuss how Gencol can be used in practice, and finally, in section 6 we touch upon related work and formulate some conclusions.

2 The problem of concept-learning

Intuitively speaking, concept-learning* (Mitchell, 1982) is the ability to generalize specific observations, by extracting common and important features characterizing these observations. A *concept* can be defined as a set of observations with common features. For each concept, there are two classes of observations: those belonging to the concept (the *positive* observations), and those not belonging to the concept (the *negative* observations). The aim of concept-learning is to find a way to distinguish positive from negative observations.

For the purposes of learning concepts with machines, suitable representations for observations and concepts are needed. Therefore, it is assumed that the observations are described (i.e., represented) in a language of observation-descriptions L_e. Descriptions of observations are *examples*. *Positive* examples correspond to observations that belong to the concept, whereas *negative* examples do not. Analoguously to examples, concepts are described in a language of concept-descriptions L_C. Rather than speaking of concepts, we will talk about concept-descriptions. Together, L_e and L_C constitute the knowledge representation formalism of examples and concepts. From now on we ignore concepts and observations and work with concept-descriptions and examples.

Obviously, the learner must be able to relate the language of concept-descriptions L_C to the language of examples L_e. So, we also assume a predicate *covers*(Cd, e), where $Cd \in L_C$, $e \in L_e$, which is true iff the observation described by the example e is in the concept described by the concept-description Cd. Roughly speaking, the truth-value of *covers*(Cd, e) denotes whether or

*Our definition of concept-learning will be very similar to Mitchell's except that we use *concept* instead of *generalization* for reasons discussed by Langley (1987), and that we make abstraction of the quality-criterion.

Problem Statement 1 *Concept-Learning*

- **Given:**
  - a language of examples L_e
  - a language of concepts L_C
  - the *covers* relation between L_C and L_e
  - a set of positive examples P
  - a set of negative examples N
  - an *Acceptance-Criterion* A
- **Find:** concept-description(s) $C \in L_C$ satisfying the *Acceptance-Criterion*

not the example e belongs to the concept represented by Cd. We will say that Cd covers e iff *covers*(Cd, e) = true. We will also use the notation *covers* (Cd) to denote the set of all examples covered by the concept-description Cd.

The aim of concept-learning is to find a concept-description in L_C which satisfies some acceptance criterion. The acceptance criterion (Mitchell, 1982; Shapiro, 1981; Michalski, 1983) often states that the concept-description must be complete and consistent.

A concept-description is *complete* w.r.t. a set of positive examples iff it covers all positive examples in the set. A concept-description is *consistent* w.r.t. a set of negative examples iff it does not cover any negative example in the set.

Complete and consistent concept-descriptions do not always exist. The reason may be that L_C is too restricted, or that the sets of positive and negative examples overlap. This last reason may also occur indirectly because of a discrepancy between observations and examples, or between concepts and concept-descriptions.

Because complete and consistent concept-descriptions do not always exist, there exist also relaxed acceptance-criteria (see, e.g. Buchanan & Mitchell, 1978 and Valiant, 1984). In general, the *Acceptance-Criterion* depends on the sets of positive and negative examples (P and N), the considered concept-description C, the concept-description language L_C and the quality criterion Q. The acceptance criterion yields either the value true or false. $Q(C,P,N)$ measures how well the description fits the examples or data. Very often $Q(C,P,N)$ is defined as the ratio of positive examples covered by C over the total number of examples covered by C. One can imagine a variety of acceptance criteria. Consider, for example, the following ones:

- *Acceptance-Criterion* (C,L_C,N,P,Q) = true iff Q reaches its maximum value in C within L_C
- *Acceptance-Criterion* (C,L_C,N,P,Q) = true iff $Q(C,P,N) > \theta$

Concept-learning is formally defined in Problem Statement 1, and illustrated in Example 1.

3 Concept-learning as search

Most of the concept-learning techniques described in the literature, can be viewed as search-techniques (cf. Mitchell, 1982), in which the states are the hypotheses in L_C and the aim is to find one or more states satisfying the acceptance criterion. With this in mind, it is easy to develop a simple generate and test algorithm, which generates all concept-descriptions, and tests whether they satisfy the quality criterion. This algorithm is known as the enumeration technique, and has been studied by Angluin & Smith (1987), Biermann (1988), and Gold (1967). Although the enumeration technique has some interesting properties it is, as for other search-problems (Angluin & Smith, 1983), more advantageous to structure the search-space.

3.1 *Structuring the search-space*

For concept-learning, the *is more specific than* relation provides a natural and powerful basis to organize the search (see Mitchell, 1982). A concept-description C_1 *is more specific than* a

Example 1: A concept-learning problem and its solution

Consider the following concept-learning problem (the taxonomies are shown in Fig. 1):

- L_e = {$x \wedge y \mid x$ is a leaf of the shape taxonomy and y is a leaf of the colour taxonomy}
- L_C = {$x \wedge y \mid x$ is a node of the shape taxonomy and y is a node of the colour taxonomy}
- *covers*($cx \wedge cy, ex \wedge ey$) = true iff cx is an ancestor of ex in the taxonomy for shape and cy is an ancestor of ey in the taxonomy for colour
- P = {square $\wedge$ red, rectangle $\wedge$ blue}
- N = {circle $\wedge$ orange, triangle $\wedge$ pink}
- The acceptance criterion is completeness and consistency with respect to the examples

This problem has several solutions, for instance: 4-sided $\wedge$ color and shape $\wedge$ mono.

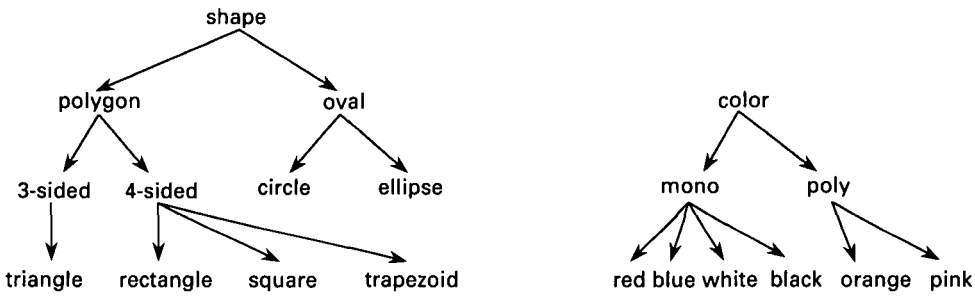

Figure 1 Taxonomies of the concept-learning problem

concept-description C_2 iff all examples covered by C_1 are also covered by C_2 (i.e., *covers*(C_1) $\subseteq$ *covers*(C_2)). We also say that C_1 is a specialization of C_2, C_2 is a generalization of C_1 or C_2 is more general than C_1. When *covers*(C_1) $\subset$ *covers*(C_2,) we will say that C_1 is a *strict* specialization of C_2. This notion of generality is illustrated in Fig 2. In example 1, we have, for instance, that oval $\wedge$ poly is (strictly) more general than circle $\wedge$ pink.

Notice that the relation *is more specific than* is transitive and reflexive. Unfortunately, it is not always anti-symmetric since there may exist several concept-descriptions which cover the same set of examples. Such descriptions are called *syntactic variants*. In Example 1, 3-sided $\wedge$ poly and triangle $\wedge$ poly are syntactic variants.

In theory, one can obtain a partial order by introducing an equivalence class and working with a canonical form as a representative of the equivalence class. In practice, there is not always a decision procedure to check whether two descriptions are syntactic variants (cf. Buntine, 1988). This sometimes leads to problems when searching the space of concept-descriptions.

It is convenient to introduce a special top element $\top$ (cf. Shapiro, 1983) and a special bottom element $\bot$ such that $\top$ covers all examples in L_e and $\bot$ does not cover any example. In example 1, we have that $\top$ = shape $\wedge$ colour, and L_C as defined in Example 1, contains no bottom element.

The relation *is more specific than* can be used to prune the search provided that the *Quality-Criterion* is well-behaved with respect to the covers relation:

A Quality-Criterion Q behaves well with respect to the covers relation iff for all concept-descriptions C_1, C_2 such that C_1 covers more positive examples than C_2 without covering more negative ones or C_1 covers fewer negative examples than C_2 without covering fewer positive ones, $Q(C_1,P,N) \geq Q(C_2,P,N)$.

The notation $Q(C_1,P,N) \geq Q(C_2,P,N)$ means that C_1 better fits the data (examples) than C_2 according to Q.

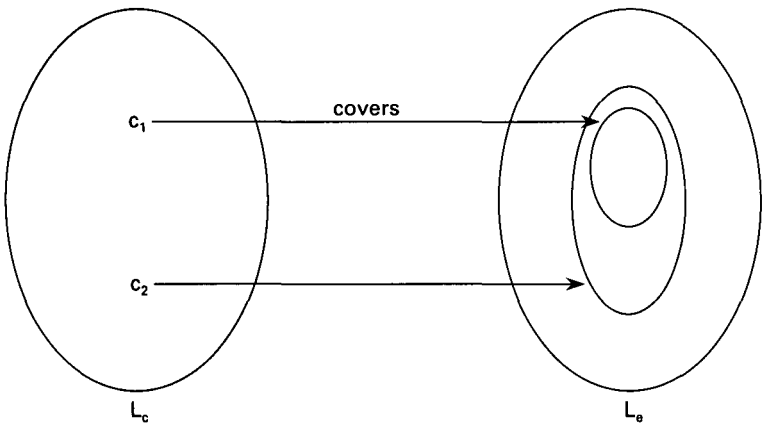

Figure 2 C_1 is more specific than C_2

When the Quality-Criterion behaves well it is a good idea to employ generalization and specialization as the basic operations to move throughout the search space of concept-descriptions. The reason for this is that when generalizing a concept-description C_1 to C_2, we have that all examples covered by C_1 will remain covered by C_2. If no additional negative examples are covered by C_2, we obtain an improvement according to our Quality-Criterion. Dually, when specializing a concept-description C_1 to C_2, we have that all examples that are not covered by C_1 will not be covered by C_2 either. The very same properties allow us to prune the search space when looking for complete and consistent concept-descriptions. For example, when the concept-description `4-sided` $\wedge$ `mono` covers the negative example `square` $\wedge$ `red`, we know that all generalizations of `4-sided` $\wedge$ `mono` will also cover the example. Therefore, when searching for consistent concept-descriptions we should only consider specializations of `4-sided` $\wedge$ `mono` such as `4-sided` $\wedge$ `white`. A dual statement can be made concerning generalization with regard to an uncovered positive example.

The conclusion of these considerations is that generalization and specialization are useful operations for concept-learning as they allow to structure the search space. Because there exist several ways to generalize and to specialize, and several possible choices for the representation languages of concepts and examples, we now introduce a framework in which these differences can easily be characterized. Using the resulting alternatives in Gencol will yield different algorithms. Applying the language-framework in conjunction with Gencol allows to characterize most concept-learning algorithms using few words.

3.2 A language framework for concept-learning

Most concept-learning systems use subsets (or variants) of first order logic (Genesereth & Nilsson, 1987) as description-languages. Without loss of generality, we assume that concept-descriptions in these systems are of the following form:

$$F_1 \vee F_2 \vee \ldots \vee F_n; \quad \text{such that all } F_i \text{ are conjunctive formulae} \quad (3.1)$$

Given concept-descriptions of this form, we need *transformers* that operate on a formula (concept-description) and return a set of generalizations or specializations of the given formula. Basically, transformers expand one node (concept-description) in the search space into a set of nodes. This is akin to general search principles (cf. Nilsson, 1980). Let us therefore introduce generalization and specialization transformers.

Generalization transformers always yield a set of generalizations of the expanded concept-description. In Example 1, the transformer that starts from a concept-description $f \wedge g$ and returns the set of all formulae $f' \wedge g'$ such that f' is a parent of f and g' is a parent of g, is a generalization transformer. *Specialization transformers* are defined dually.

It turns out that there exist several special kinds of generalization and specialization transformers. We discuss some of the most important ones.

Specializations or generalizations of formulae of the form (3.1), can be obtained by specializing or generalizing individual disjuncts of the formula. For example, the concept-description (`triangle` $\wedge$ `red`) $\vee$ (`4-sided` $\wedge$ `blue`) can be generalized by applying a generalization transformer on the first part. This could yield, for example, (`triangle` $\wedge$ `mono`) $\vee$ (`4-sided` $\wedge$ `blue`). Transformers that operate in this way, are called *derived* transformers. The corresponding transformers on the individual disjuncts are called *formula* transformers.

A particularly important class of transformers is the class of transformers that always yields minimal changes to the concept-description. These transformers are called *operators*. A generalization (respectively specialization) operator is a generalization transformer that maps a concept-description on its minimal (strict) generalizations (respectively maximal (strict) specializations) in the concept-description language. Considering the above concept-learning problem, a specialization operator can be obtained by mapping all concept-descriptions $f \wedge g$ onto the set of all formulae $f' \wedge g$ and $f \wedge g'$ where f' and g' are children of f and g, respectively.

Example driven transformers take into account a particular example when generalizing or specializing. Example driven generalization transformers use a positive example. They return only concept-descriptions that cover the example. Example driven specialization transformers use a negative example and return only descriptions that do not cover the example. In our example problem, we can define the following example driven generalization operator. Map a description $f_1 \wedge g_1$ and an example $f_2 \wedge g_2$ on the set of all descriptions $f_3 \wedge g_3$ such that f_3 is the least common ancestor of f_1 and f_2 and g_3 is the least common ancestor of g_1 and g_2.

3.3 The GENeric COncept-Learning algorithm

Below, a detailed description of the generic algorithm Gencol is presented. Possible instantiations for the parameters of Gencol are briefly discussed in section 3.4. Variables of Gencol are in roman and parameters in *italic*. It is assumed that all procedures (i.e., generic parameters) of Gencol have access to P,N,L_e,L_C and the Memory.

Partial Solutions is used as a queue which contains the concept-descriptions that are currently under consideration. Complete Solutions contains the output of Gencol. The Memory is (sometimes) used to keep track of concept-descriptions that cannot lead to a solution. P is the set of positive examples and N the set of negative examples. ps is a variable which contains the concept-description (partial solution) that is currently under consideration and that will be expanded or moved into the set of complete solutions. In ex, Gencol may store an example to be used by an example driven transformer. *Optional* specifies that the ex := *Generate Example* part is only invoked under specific conditions. *Oneof* **[spec, gen]** means that either the **spec** or the **gen** branch is invoked.

3.4 Possible choices for Gencol's parameters

In this section, we indicate how Gencol can be instantiated along several dimensions, in order to familiarize the reader with the algorithm and with the principles of concept-learning.

3.4.1 *Languages and transformers*

The choice of the languages and transformers is a very important one. The languages determine the knowledge representation of the problem and the form of the search space. The transformers specify the basic steps which the system can take in this search-space. When comparing concept-learning with graph-searching, the nodes are the hypotheses in the language of concepts L_C and the arcs are determined by the transformers. In Gencol, we can use any language and transformer. Of course, different choices give rise to different algorithms.

Algorithm 1: Gencol

```
procedure Gencol
    Partial Solutions := Initialize
    Complete Solutions := ∅
    Memory := ∅
    while not Stop-Criterion (Partial Solutions, Complete Solutions) do
        ps := Select (Partial Solutions)
        Partial Solutions := Partial Solutions − {ps}
        if Acceptance-Criterion (ps) is satisfied
        then Complete Solutions := Complete Solutions ∪ {ps}
        else if Optional
            then ex: = Generate Example (ps, Partial Solutions, Complete Solutions)
            endif
            Oneof [
                spec: Partial Solutions :=
                  Partial Solutions ∪ Specialization-transformer (ps,ex)
                gen: Partial Solutions :=
                  Partial Solutions ∪ Generalization-transformer (ps,ex)
                ]
            Memory := Memory ∪ Remember (ps)
        endif
        Memory := Adapt (Memory)
        Partial Solutions := Filter (Partial Solutions)
    endwhile
endproc
```

3.4.2 *The initialization*

The function *Initialize* determines the starting point of the search for a concept. The initialization may yield one or more initial concept-descriptions. Systems that start at $\top$ and only specialize (e.g., the Model Inference System (Shapiro, 1983)) are often called *general-to-specific* systems. The dual kind of system, searching by generalizing from $\bot$, is known as a *specific-to-general system* (cf., e.g., Clint (De Raedt & Bruynooghe, 1992)).

3.4.3 *The Stop-criterion*

The *Stop-Criterion* specifies the conditions which terminate the search. By defining it appropriately it is possible to obtain one (perhaps the best), a few or all solutions.

3.4.4 *The Acceptance-criterion*

The *Acceptance-Criterion* characterizes complete solutions to the concept learning problem. If a partial solution satisfies these conditions it will be moved to the set of complete solutions which contains the result of the algorithm. Acceptance-criteria that can be used in Gencol include completeness and/or consistency (Mitchell, 1982), the best hypothesis according to some Lexicographic Evaluation Function (LEF) (Michalski, 1983), achieving at least a specified score and coverage on the examples (Buchanan & Mitchell, 1978), heuristically significant and most compact concept-descriptions (Quinlan, 1986), maximally general or maximally specific concept-descriptions (Dietterich & Michalski, 1983) and probably approximate correct concept-descriptions (Valiant, 1984).

The only requirement on the acceptance-criterion is that the underlying quality criterion is well-behaved with respect to the covers relation. Nevertheless, this requirement is an important one as it characterizes a necessary precondition for successfully using any algorithm that employs the

Gencol schema. Gencol can also be seen as a hill-climber. In this context, the requirement means that there should be no local maxima but rather smooth tops, which explains why Gencol does not make abstraction of genetic learning approaches.

The above observation is double-edged. On the one hand, it suggests possibilities for generalizing existing algorithms and on the other hand, it represents a limitation of most existing methods. Related observations were formulated by Langley *et al.*, (1987) and Rendell (1986).

3.4.5 *The search-strategy*

According to the function *Select* one obtains different search-strategies (cfr. Nilsson, 1980): when *Select* is first-in-first-out Gencol performs a breadth-first search, for last-in-first-out a depth-first search and for maximization (or minimization) of a heuristic evaluation function (satisfying certain criteria; cf. (Nilsson, 1980)) a best-first search; splitting Partial Solutions into two dual versions results in a bidirectional search as in version spaces, etc.

Another aspect of the search-strategy concerns the choice of the transformation to apply (i.e., the definition of *Oneof*). Some possibilities include: only generalization or specialization, preferring generalization over specialization (or *vice versa*), or a mixture of both. In many so-called data-driven algorithms *Oneof* depends on whether a negative example is covered or a positive example is not covered. In the former case, the concept-description is specialized while in the latter it is generalized.

3.4.6 *Using heuristics*

As the search space may be very large, it is often desirable to guide the search by means of domain-knowledge and heuristics. Such guidance can be built in Gencol in a variety of ways:

- *Select*: can use a heuristic evaluation function, selecting the most interesting partial solutions first.
- *Filter*: can be used to retain only the most promising partial solutions. If only a fixed number of partial solutions is retained, then a beam-search is realised (e.g., Michalski, 1983). *Filter* can also be used to discard partial solutions which contradict a domain-theory (e.g., Buchanan & Mitchell, 1978; Subramanian & Feigenbaum, 1986).
- The transformers may also use knowledge or heuristics. Such heuristics often deal with the selection of the formula to be replaced when applying a derived transformer. For instance, when a concept-description is generalized in order to cover an example, this can be done by generalizing one of the formulas of the description. According to conceptual distance measures (Michalski & Stepp, 1983; Michalski, 1987) one may identify the best formula to be replaced.

3.4.7 *The memory*

Some systems, like the Model Inference System (Shapiro, 1981, 1983) store incorrect concept-descriptions in order to avoid infinite loops and considering unpromising concept-descriptions more than once. Others, like, e.g., Quinlan (1986) and Michalski (1983) do not keep track of incorrect concept-descriptions. This behaviour can be realized by defining *Remember* and *Adapt* appropriately.

3.4.8 *Examples*

P and *N* are the sets of positive and negative examples. They may be given at the start or acquired incrementally. When they are acquired incrementally, the system and/or the user may generate additional examples at certain moments in time. *P* and/or *N* then have to be updated (this is not shown explicitly in the algorithm as it is trivial). *Generate Example* may be used for the purpose of letting the system or the user generate an example. *Optional* then states the conditions under which this is done. However, *Generate Example* is also used in connection with example-driven transformers. In this case, it initializes the example ex which is used in the example-driven

transformers. The conditions are again specified by *Optional*. If the transformers are not example-driven then the value of ex is **void**.

4 Well-known instantiations of Gencol

In this section we indicate how Gencol can be instantiated to obtain a wide variety of systems. To the best of our knowledge, all algorithms that have a well-behaved quality criterion underlying the *Acceptance-Criterion*, and that search the space by applying generalization and specialization can be obtained. Evidence for this conjecture has been provided (De Raedt, 1991) by showing that Gencol instantiates to the following seemingly disparate algorithms: the TDIDT-family (Quinlan, 1986), the Model Inference System (Shapiro, 1981), the AQ methodology (Michalski, 1983) and the version space algorithm (Mitchell, 1977). These systems were chosen because they cover a wide range of concept-learning approaches, and because each of them is thoroughly described, well-known, and has been successfully applied. Due to space restrictions and the technicality of the instantiations of Gencol to MIS and the version space algorithm, we do not present these instantiations here. The interested reader is referred elsewhere (De Raedt, 1991) for a presentation of these systems in the context of Gencol. Throughout the presentation of instantiations of Gencol, we take the freedom to omit technical details of minor importance. Furthermore, we only discuss these systems as instantiations of Gencol. A discussion of all aspects of these systems is outside the scope of this paper. For more details we refer to the original papers and to some recent overview papers (Kalkanis & Conroy, 1991; Kocabas, 1991).

4.1 Typical concept-learning systems

Typical concept-learning systems, like, e.g. Rulegen and Rulemod (Buchanan & Mitchell, 1978), Sprouter (Hayes-Roth & McDermott, 1978), Thoth (Vere, 1975) Marvin (Sammut & Banerji, 1986), Clint (De Raedt & Bruynooghe, 1992; De Raedt, 1992) and the Algorithms in Laird (1986) and Valiant (1984) have a similar structure in the Gencol framework.

- they process one example at a time (all but Rulegen and Rulemod).
- they work general-to-specific (Rulegen) or specific-to-general (Sprouter, Thoth, Valiant's A and B algorithms, Laird (1988) and Clint) or apply both specialization and generalization (Marvin and Rulemod).
- most of the systems search breadth-first (all except Marvin, which uses depth-first search)
- they all use a variant of first order logic (all except Rulegen and Rulemod) in which an example corresponds to a conjunction and a concept-description to a disjunction of conjunctions or a conjunction.
- the transformers in most of these systems are example-driven operators (all except Rulegen and Rulemod)
- some systems like Marvin, Clint and Valiant's B algorithm generate their own examples
- in some systems all possible partial solutions are kept (as in Laird, 1986) while in Rulegen, Rulemod and Sprouter only the best ones are kept.
- also the acceptance criterion differs: in Sprouter, Thoth, Laird (1986), Clint and Marvin it is completeness and consistency, while in Valiant's A and B algorithms it is pac-learning and in Rulegen and Rulemod local maximization of a heuristic evaluation function

4.2 *The Top-Down Induction of Decision Trees family*

The TDIDT-family of algorithms (Quinlan, 1986) induces decision trees from a given set of positive and negative examples, There are many variants of the basic TDIDT algorithm, but most of the differences have to do with the heuristics employed (cf. below). Examples consist of attribute-

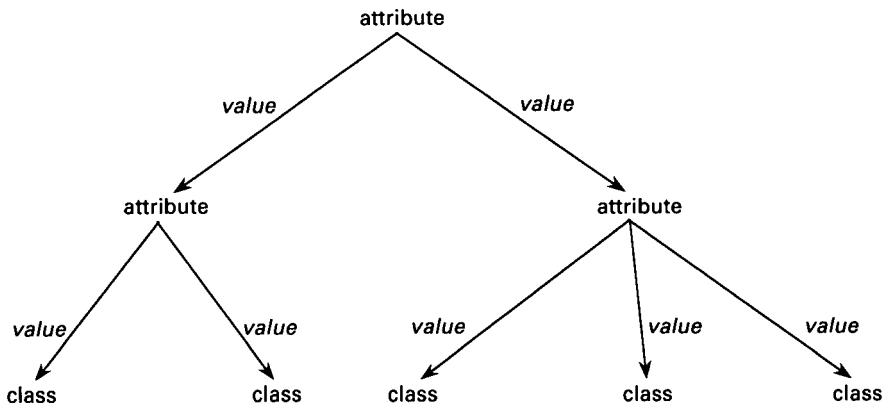

Figure 3 Structure of a decision tree

value pairs. A decision tree is shown in Fig. 3. It has attribute-based tests as nodes and for each possible outcome of the test there is a branch in that node. Leaves of the tree correspond to classes. In order to classify an example using a decision tree, one starts in the root of the tree and always takes the branch corresponding to the value for the attribute in the example, until one arrives at a leaf. The TDIDT-family induces trees by the following simple algorithm. It starts with all examples in the root of the tree. There it selects the best attribute to test upon according to statistical heuristics and then it constructs a branch for each value of the attribute* and propagates all examples with that value to the node on the corresponding branch. If the node and the examples in it satisfy a *node-acceptance-criterion*, the node becomes a leaf with as label the class assigned by the node-acceptance-criterion. The node-acceptance-criterion states for instance that all examples in the node have to belong to the same class; this class is then taken as the label. Otherwise, the process is recursively applied on that node with the propagated examples. Nodes which still have to be handled will be called partial nodes. Nodes which are already handled are completed nodes.

The TDIDT-family of algorithms can easily be realised by Gencol. One only has to instantiate Gencol as follows:

- L_C = the set of decision trees
- L_e = $\{e|e$ is a conjunction of attribute value pairs$\}$
- P and N are given initially and fixed.
- *Acceptance-Criterion* (ps) = true iff all nodes in ps are completed
  The completion of all nodes implies that all leaves must satisfy the *Node-Acceptance-Criterion*. The *Node-acceptance-criterion* states, e.g., that all examples in the node must belong to the same class (this results in complete and consistent decision trees) or that the node cannot be expanded in a statistically significant way (see Quinlan, 1986).
- *Stop-Criterion* (Partial Solutions, Complete Solutions) = |Complete Solutions| = 1
- *Select* ({ps}) = ps
  Each time *Select* is called, the Partial Solutions will be a singleton.
- *Oneof* (**spec, gen**) = **spec**
  The algorithm only specializes.
- *Optional* = false
  This implies that ex and *Generate example* are void.
- *Initialize* = { ⊤ }
  ⊤ is the tree consisting of one partial node
- *Generalization-transformer* (ps) = void

*Numerical attributes are treated differently here. Usually, they are split up in intervals and the tests succeed when the value falls in the interval. Some members of the TDIDT family such as ASSISTANT (Cestnik et al., 1987) always group different values for one attribute together.

- *Specialization-transformer* (ps) =

```
select a partial node n of ps
if n satisfies the Node-acceptance-criterion
then turn n into a leaf by assigning a class to it
     n becomes completed
else for all attributes a that do not occur on the path from the root of ps to n do
          expand the node n using attribute a
          return the set of all these expansions
     endfor
endif
```

  Expanding a partial node using an attribute a is done by testing on attribute a in that node and making a branch for each value of the attribute. The node n is completed and the new nodes are partial ones. The *Specialization-transformer* will expand one node of the tree. It will yield as many new trees as there are attributes which do not yet occur on the selected branch. A specialized tree tests on the value of one more attribute. It is not important which branch of the tree is selected for expansion, as every possible choice will eventually yield the same result.
  Technical note: in order to obtain a real specialization C of a tree C', in the sense that *covers* $(C) \subseteq$ *covers*(C'), it is also necessary to define *covers* for trees that contain incomplete nodes. The above specified transformer computes real specializations when *covers* is defined as follows *covers*$(C) = \{ e \in L_e |$ when classifying e using C, e is not assigned the label *negative* $\}$; C may be an incomplete tree: nodes that are incomplete are treated as if they were *positive* leaves by default.
- *Remember* (ps) = $\varnothing$
- *Adapt* (Memory) = $\varnothing$
  So, nothing is remembered.
- *Filter* (Partial Solutions) = {ps}, where ps is the best decision tree according to a heuristic that measures the heuristic gain of expanding a node of the tree. Several heuristics have been proposed (cf. Gams & Lavzač, 1987; Quinlan, 1986). Different heuristics result in different algorithms. The above instantiation of Gencol simulates the basic TDIDT-algorithm and the extension for pre-pruning (Cestnik *et al.*, 1987). It can also be extended to handle post-pruning (Cestnik *et al*, 1987) and to simulate the production rule derivation method of Quinlan (1987).

4.3 *The AQ methodology using Induce*

Michalski (1983) describes the AQ concept-learning algorithm. It uses the Induce program (Michalski, 1983) as a subroutine. It has been used to learn diagnostic rules for soy-bean diseases (Michalski & Chilausky, 1980). Although there are many versions of the system, we consider that described by Michalski (1985) pp 112–115, and present only a simplification without truth-preserving transformations or constructive induction. These features are omitted since they are not essential to the algorithm as an instantiation of Gencol, and presenting these features would make the presentation less readable and understandable.

The AQ algorithm learns a complete and consistent concept-description from a given (and fixed) set of positive and negative examples. Concepts are represented in Annotated Predicate Calculus, which is a variable-valued logic language (Michalski, 1975). For our purposes, it suffices to regard them as variants of first order logic in which concepts are represented by a disjunction of conjunctions and examples by conjunctions. AQ learns as follows: it selects a positive example and invokes Induce in order to generate a number of maximally general formulas which cover the positive example and which do not cover negative examples; then it selects the best formula in the result of Induce according to a lexicographic evaluation function (LEF), and adds it as a disjunct to the current concept-description; if the current concept-description is complete AQ stops; otherwise it generates a positive example that is not covered by the current concept-description and restarts.

Induce learns as follows: it first uses the positive example to define a specialization-operator, which refines by adding conjuncts to a formula. The conjuncts are added in a heuristic order (the most interesting first) and only conjuncts which are of lower preference than conjuncts already in the formula are added (this to avoid generating the same formulas more than once); the algorithm starts at $\top$ and refines those formulas which are not consistent. Also, all consistent concept-descriptions are locally generalized in order to be as general as possible, while remaining consistent. During the search Induce only keeps track of the m best formulas and searches for n_1 complete and consistent solutions or for n_2 consistent solutions. m, n_1 and n_2 are parameters of Induce.

In order to obtain this algorithm in Gencol, it is convenient to define also two different algorithms in Gencol: one for Induce and one for AQ*.

4.3.1 The AQ method

- L_C = Annotated Predicate Calculus (Michalski, 1983).
- L_e = the conjunctive statements (c-expressions) in Annotated Predicate Calculus (Michalski, 1983).
- P and N are given initially and fixed.
- *Acceptance-Criterion* (ps) = true iff ps is complete and consistent.
- *Stop-Criterion* (Partial Solutions, |Complete Solutions|) = Complete Solutions = 1
- *Select* ({ps}) = ps
  each time *Select* is called, Partial Solutions is a singleton.
- *Oneof* (**spec,gen**) = **gen**
  AQ always generalizes.
- *Optional* = true.
- *Generate Example* (Partial Solutions, Complete Solutions)) yields a positive example which is not covered by ps.
- *Initialize* = $\{\bot\}$.
- *Generalization-transformer* (ps,ex) = Induce is called and {ps v f f is a formula returned by Induce} is returned.
  So, the *Generalization-transformer* is derived from Induce (a formula-transformer).
- *Specialization-transformer* (ps,ex) = void.
- *Remember* (ps) = $\emptyset$
- *Adapt* (Memory) = $\emptyset$
  So nothing is remembered.
- *Filter* (Partial Solutions) = {ps} where ps is the best element of Partial Solutions according to the LEF.

4.3.2. *The Induce method*

- L_C = a subset of the Annotated Predicate Calculus without external disjunction (Michalski, 1983).
- L_e = the conjunctive statements (c-expressions) in Annotated Predicate Calculus (Michalski, 1983).
- P and N are given initially and fixed; P is a singleton $\{p\}$.
- *Acceptance-Criterion* (ps) = true iff ps is consistent and locally maximal, i.e. it cannot be generalized without becoming inconsistent.
- *Stop-Criterion* (Partial Solutions, Complete Solutions) = true iff there are n_1 complete and consistent solutions on Complete Solutions or there are n_2 consistent solutions on Complete Solutions.
- *Select* (Partial Solutions) = best first according to some heuristic.
- *Oneof*(**spec,gen**) =

*One could also present AQ using one instantiation of Gencol but this would be less readable.

is ps **is consistent**
then **gen** (*in order to find a local optimum*)
else **spec** (*in order to find a consistent description*)
endif

- *Optional* = false
  This implies that ex and *Generate Example* are void.
- *Initialize* = $\{\top\}$.
- *Generalization-transformer* (ps,ex): some of Michalski's (1983) generalization rules are applied.
- *Specialization-transformer* (ps,ex) = $\{ ps \wedge c \mid c$ is a literal in ex of lower preference than predicates already occurring in ps$\}$.
- *Remember* (ps) = $\emptyset$
  Adapt (Memory) = $\emptyset$
  So, nothing is remembered.
- *Filter* (Partial Solutions) = the set of m best elements of Partial Solutions

  In this way a beam-search is realized. The best formulas are those that score best on the LEF. A typical LEF is the maximization of the number of covered positive examples and minimization of the complexity of the expression.

It turns out that many algorithms use a variant of the AQ algorithm to learn disjunctive concepts. For instance the Model Inference System (Shapiro, 1983), Clint (De Raedt & Bruynooghe, 1992; De Raedt, 1992) and Marvin (Sammut & Benerji, 1986) can be seen as variants of AQ: indeed, they all learn conjunctive formulas by invoking a special algorithm. In the Model Inference System, this is done by refining from $\top$ and in Marvin and Clint by generalizing from $\bot$. The result of the special algorithm is then added as a disjunct to the current concept-description.

5 Discussion of Gencol

In the previous section we demonstrated that Gencol can be instantiated in order to obtain a wide variety of concept-learning algorithms. In this section, we indicate how to use Gencol to realize systems that employ background knowledge and constructive induction, and we discuss how Gencol can be used by practitioners of artificial intelligence and knowledge engineering.

5.1 Background knowledge and constructive induction

Recently, two novel trends emerged in concept-learning. One is to employ large bodies of background knowledge during the learning. Typical systems falling into this category include Cigol (Muggleton & Buntine, 1988), Clint (De Raedt & Bruynooghe, 1992), Blip (Morik, 1989; Wrobel, 1989), Foil (Quinlan, 1990) and Linus (Lavrač *et al.*, 1991). The other trend aims at extending the vocabulary of the concept-learner while learning a particular concept, i.e., constructive induction (Matheus, 1989) or predicate invention (Muggleton & Buntine, 1988). This may be necessary or desirable when the available vocabulary proves to be inadequate for the concept-learning task at hand. For example, in an attribute value framework one might introduce a new attribute that is defined in terms of the available ones. Using this newly defined attribute may result in better concept-descriptions. Typical approaches that realize a form of constructive induction or predicate invention include Muggleton & Buntine (1988), Wrobel (1988), Mehra *et al.*, (1989) and Matheus (1989).

Clearly, Gencol can be used in the context of constructive induction and background knowledge without any difficulty. The reasons for this are summarized below:

- The difference between classical concept-learning and concept-learning using constructive induction lies in the transformations. To introduce a new concept, attribute or predicate, one

needs special transformations. The main problem in constructive induction is to design heuristics and strategies to control the application of these transformations. To model concept-learners using constructive induction in Gencol, one only needs to define these special transformations, their applicability conditions and their evaluation criteria. Within Gencol, this basically amounts to formalizing the transformations, the *Oneof* function and the *Filter*.

- When using background knowledge in concept-learning, two issues change: the transformations being applied, and the representation of concepts. The semantics of a concept-description in this context depends on the available background theory. Because the learning algorithm may change the background theory, in the context of Gencol it is necessary to define concept-descriptions as background theory + concept-definition, i.e., a theory. Transformations then transform one such theory into another one. Seen in this way, all transformations typically employed in the presence of background knowledge (e.g., abduction, induction, inverse resolution, unfolding, etc.), also apply in the context of Gencol.

To illustrate the fact that Gencol also applies in the context of background theory and constructive induction, let us briefly indicate how the system Cigol (Muggleton & Buntine, 1988) (which uses both background knowledge and constructive induction) is an instantiation of Gencol. In Cigol, a concept-description corresponds to a logical theory represented as a Horn-clause program. The transformations employed by Cigol are based on inverse resolution. Inverse resolution transforms one theory (Horn-clause program) into a more general one. The search strategy is a best first one guided by information compression. Given these observations, it is easy to see that Cigol (and therefore also background theory and constructive induction) fit into Gencol.

5.2 *Applying Gencol in practice*

5.2.1 *A tool derived from Gencol*

As part of the Masters thesis of Erwin Declercq (1990), a tool to aid in the design of concept-learning systems was developed. The tool was based on a straightforward implementation of Gencol in Prolog. To obtain a real tool, facilities were provided to choose the generic parameters in a declarative way:

- A logic language based on Prolog was used as the representation mechanism for L_e and L_C. Different classes of description languages can be obtained by appropriately choosing certain parameters. Indeed, it is possible to state whether a concept-description:
  - may be disjunctive or conjunctive
  - may use tail-recursion, recursion or no recursion
  - may use local existentially quantified variables or not (cf. De Raedt, 1992)
  - may use a form of explicit negation as in De Raedt (1992)
  - may use functors and if so, which ones.
- Once appropriate languages are defined, the user can specify which transformer(s) should be used. The user only needs to declare the kind of transformer she wants to use. For example, she can state that the generalization transformer should be an example driven operator. The available transformers correspond basically to the ones discussed above.
- The tool can cope with mode and type information concerning predicates that may be used in concept-description languages. This allows us to optimize the search.
- The system uses a depth-bounded interpreter in order not to run into infinite loops while verifying the coverage of certain examples.
- The search strategy can be chosen: depth-first, breadth-first, beam-search or best-first. In the last two cases a lexicographic evaluation function (Michalski, 1988) is used to identify the best elements. A default LEF is built-in but can easily be modified.

- The conditions under which to specialize, to generalize, to discard a description from the list of partial solutions (*Filter*), to satisfy the stop and acceptance criterion, can be assembled using the following primitives:

  – consistent
  – complete
  – maximally general
  – maximally specific
  – the best ones according to the LEF
  – minimally disjunctive
  – covers (resp. does not cover) a positive (resp. negative) example; the example is a parameter, e.g. **ex**.

  The primitives can be used in conjunction with each other. It is, e.g., possible to state that all descriptions that are not minimally disjunctive, or cover the negative example **ex** have to be discarded from the list of partial solutions. The formalism to specify these conditions is quite friendly to the user.

When the builtin formalism and/or facilities are not sufficient to specify a desired algorithm, the user can easily modify certain parameters by creating **Prolog** code for these parameters.

The tool has been used to obtain a simple implementation of the algorithms discussed above: version spaces, **AQ**, and a simple version of the **Model Inference System** and **TDIDT**. This provides clear evidence for the claim that **Gencol** can be instantiated to these algorithms and for the generality and usefulness of **Gencol** as a framework for concept-learning.

5.2.2 *Using Gencol as an implementation schema*

The tool based on **Gencol** is only one way in which **Gencol** is relevant to the practice of knowledge engineering. **Gencol** can also be used as an implementation schema for designing a particular concept-learning algorithm for a particular application. The advantage of using **Gencol** in this manner is two fold. First, all the design choices are made explicit. Secondly and more importantly, when the implementation is based on **Gencol** it is very easy to experiment with variants of the algorithm. In order to obtain a variant of the implemented algorithm, one only needs to vary one of the generic parameters of **Gencol**. Experimenting using **Gencol**'s architecture is beneficial for practical applications, as there exist several possibilities for instantiating a parameter and their effect on the overall performance is often unclear.

Gencol has proven useful in this respect when developing two concept-learners at our department: **Swan** (Sablon *et al.*, 1991; Ade *et al.*, 1991) and **Clint** (De Raedt, 1992; De Raedt & Bruynooghe, 1992). In the **Swan** system, where the main novelty was the use of a deductive operator (unfolding) together with an inductive one (absorption or inverse resolution (Muggleton & Buntine, 1988)). With **Gencol** it was quite easy to turn this idea into a running system. We only had to specify when the transformers were to be invoked as the main loop of **Gencol** was taken over. Using **Gencol** also allowed to play with the search-strategy and to determine a suitable one.

6 Related work

Gencol builds further on the work of Shapiro (1981, 1983), Laird (1988), Mitchell (1982), Gams & Lavrač, 1987) and De Raedt *et al.* (1987) by gradually abstracting more features of concept-learning.

The presented work is related to all surveys of concept-learning. This includes: Biermann (1986), Mitchell (1982), Kodratoff (1988), Michalski (1983), Angluin & Smith (1983), Shapiro (1981, 1983), Laird (1986), Dietterich & Michalski (1983), Cohen & Feigenbaum (1981), Rendell (1986), Gams & Lavrač (1987), Bundy *et al.* (1985), Kalkanis & Conroy (1991) and Kocabas (1991). However the focus of all these approaches is different: Cohen & Feigenbaum (1981),

Biermann (1986) and Kalkanis & Conroy (1991) give a didactical survey of the field, Dietterich & Michalski (1983), Bundy *et al.* (1985) and Angluin & Smith (1983) compare different approaches, Rendell (1988) and Michalski (1983) present general frameworks and theories for concept-learning. Michalski (1983) also presents the AQ program and the other papers by Gams & Lavrač (1987), Shapiro (1981, 1983), Laird (1988) and Mitchell (1982) extract underlying principles of concept-learning and attempt to formulate a general algorithm that accounts for these principles. Shapiro presents his Model Inference System and demonstrates it at work on two different concept-learning tasks: he makes abstraction of the description languages and the operators; Laird focuses on a semantic theory of concept-learning, which allows him to formulate a general bottom-up system which is roughly equivalent to a version of Gencol, that starts from $\perp$ and uses only generalization by means of a generalization operator; and Gams and Lavrač implemented a practical algorithm in order to be able to test different heuristic criteria for concept-learning: their algorithm can make abstraction of the AQ and TDIDT family since it is model-driven; Mitchell compares and characterizes different algorithms in terms of their search-strategy: he defines an algorithm that searches breadth-first, one that searches depth-first and also the version space method.

Gencol goes significantly further than the approaches mentioned above, since it makes abstraction of the whole spectrum of concept-learning algorithms that use generalization and specialization, rather than just a part of that spectrum. Moreover, Gencol is more detailed than the general algorithms given by Shapiro, Laird and Gams and Lavrač. As opposed to Shapiro, Laird and Mitchell, Gencol also allows for acceptance-criteria which are different from completeness and consistency. The three algorithms in Mitchell (1982) are combined into a more general one, that can be instantiated to algorithms that are not instantiations of Mitchell's ones (e.g., the AQ program). Similarly, Gencol can be instantiated to obtain systems which cannot be obtained with the Model Inference System, or the algorithms of Laird and Gams and Lavrač. Laird's algorithm does not allow the use of generalization as well as specialization operators; Gams and Lavrač's algorithm assumes that all examples are given initially, that learning starts at $\perp$, and that the languages only allow attribute-value pairs; and the Model Inference System is an incremental algorithm, and its search-strategy uses a set of Partial Solutions, whose elements are identical except for one formula because the transformer is a derived one. Also it assumes that the *Generalization transformer* is the one derived from the null formula operator, which maps all formula on the $\top$ element.

Antagonistic to the Gencol framework is the approach taken in the Machine Learning Toolbox ESPRIT project (Causse *et al.* 1990) and related ones such as Tcheng *et al.* 1989. The Machine Learning Toolbox contains a variety of machine learning programs and is not restricted to concept-learning only. The toolbox idea is useful for practical applications as one can run several algorithms on a particular problem, evaluate their results and use the results of the algorithm that scored best. Also, one may want to first use one learning technique and feed the results of that learning step into a second learning module, etc. The key difference between the Machine Learning Toolbox and Gencol is that the Toolbox offers an environment where several *existing* concepts-learning techniques can be applied on particular problems whereas Gencol provides an environment to support the development of *novel* concept-learning techniques, possibly taking into account specific characteristics of the domain.

7 Conclusions

It has been shown that the formulation of Gencol is an interesting contribution to the theory and practice of concept-learning since it provides a unifying framework for concept-learning algorithms. Gencol builds further on Mitchell's *Generalization as search* paradigm as it makes abstraction of this search-strategy.

In contrast to most previous surveys of concept-learning, we have synthesized a wide variety of concept-learning systems into Gencol. Gencol does not only serve synthetic or didactic purposes,

but also analytic and practical ones. It is useful for analysis, because it offers a general framework in which concept-learning systems can be compared and evaluated. Also, by reformulating systems in Gencol-terms, one can easily appreciate the differences and similarities between different approaches. Furthermore, Gencol forms the basis of a tool that aids in the design of novel concept-learning algorithms or applications. One only has to choose Gencol's parameters appropriately w.r.t. the application. In this way new and interesting systems can be derived from Gencol.

Acknowledgements

We would like to thank Gunther Sablon, John Gallagher, Leon Sterling, Francesco Bergadano, Stefan Wrobel, Bruno Krekels, Dirk Van Meir, Bern Martens, Mahendra Vermaut, Mathias Gams and the referees for useful suggestions, fruitful discussions and helpful comments on this work in various stages. The authors are supported by the Belgian National Fund for Scientific Research.

References

Adé, H, De Raedt, L and Bruynooghe, M, 1992. "Inverse resolution in an integrated inductive-deductive learning system" In: *Proceedings of the 10th European Conference on Artificial Intelligence*, Wiley.

Angluin, D and Smith, CS, 1983. "Inductive inference: theory and methods", *ACM Computing Surveys* **15** 237–269.

Biermann, A, 1986, "Fundamental mechanisms in machine learning and inductive inference" In: W Bibel and P Jorrand, editors, *Fundamentals of Artificial Intelligence* Springer-Verlag.

Bratko, I, Mozetic, I and Lavrač, N, 1989. *Kardio: a study in deep and qualitative knowledge for expert systems*, MIT Press.

Buchanan, BG and Mitchell, TM, 1978. "Model-directed learning of production rules" In: DA Waterman and F Hayes-Roth, editors, *Pattern-directed inference systems* Academic Press.

Bundy, A, Silver, B and Plummer, D, 1985. "An analytical comparison of some rule-learning programs" *Artificial Intelligence* **27** 137–181.

Buntine, W, 1988, "Generalized subsumption and its application to induction and redundancy" *Artificial Intelligence* **36** 375–399.

Causse, K, Morik, K, Sims, P and Rouveirol, C, 1990. "A comparative study of the representation languages used in the Machine Learning Toolbox" In: ESPRIT 90, Kluer.

Cestnik, B, Kononenko, I and Bratko, I, 1987. "Assistant 86: a knowledge-elicitation tool for sophisticated users" In *Proceedings of the 2nd European Working Session on Learning*, pp 31–45, Sigma Press.

Cohen, PR and Feigenbaum, EA, editors, 1981. *The handbook of artificial intelligence* vol. 3, Morgan Kaufmann.

Declercq, E, 1990. Een uitwerking van het generisch algoritme Gencol dat concepten leert, Master's thesis, Department of Computer Science, Katholieke Universiteit Leuven (in Dutch).

De Raedt, L, 1991. *Interactive Concept-Learning* PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven.

De Raedt, L, 1992. *Interactive Theory Revision: an inductive logic programming approach*, Academic Press.

De Raedt, L and Bruynooghe, M, 1992. "Interactive concept-learning and constructive induction by analogy" *Machine Learning* **8** 107–150.

De Raedt, L, Krekels, B, Bruynooghe, M and Van Meir, D, 1987. "Using Shapiro's model inference system for concept-learning". In: *Proceedings of the 4th international conference on Artificial Intelligence and Control Systems of Robots*, pp 191–196, North Holland.

Dietterich, TG and Michalski, RS, 1983. "A comparative review of selected methods for learning from examples". In: RS Michalski, JG Carbonell and TM Mitchell, editors, *Machine Learning: an artificial intelligence approach*, vol. 1, Tioga Publishing Co.

Gams, M and Lavrač, N, 1987. "Review of five empirical learning systems within a proposed schemata". In *Proceedings of the 2nd European Working Session on Learning* pp 46–66, Sigma Press.

Genesereth, M and Nilsson, N, *Logical foundations of artificial intelligence*, Morgan Kaufmann.

Gold, EM, 1987. "Language identification in the limit", *Information and control* **10** 447–474.

Hayes-Roth, F and McDermott, J, 1978. An interference matching technique for inducing abstractions", *Communications of the ACM* **21** 401–410.

Kalkanis, G and Conroy, GV, 1991. "Principles of induction and approaches to attribute based induction", *Knowledge Engineering Review* **6**.

Kocabas, S, 1991, "A review of learning" *Knowledge Engineering Review* **6**.

Kodratoff, Y and Michalski, RS , editors, 1990. *Machine Learning: an artificial intelligence approach, Vol.* 3, Morgan Kaufmann.

Kodratoff, Y, 1988. *Introduction to Machine Learning* Pitman.

Laird, PD, 1988. "Inductive inference by refinement". In: *Proceedings of the 5th American Conference on Artificial Intelligence* pp 472–476, Morgan.

Langley, P, 1987. "The terminology of machine learning" *Machine Learning* **1** 141–144.

Langley, P and Carbonell, JG, 1987. "Machine learning". In: S Shapiro, editor, *Encyclopedia of artificial intelligence*, Wiley.

Langley, P, Gennari, JH, and Iba W, 1987. "Hill-climbing theories of learning". In: *Proceedings of the 4th International Workshop on Machine Learning* pp 312–323, Morgan Kaufmann.

Lavrač, N, Džeroski, S and Grobelnik, M, 1991, "Learning non-recursive definitions of relations with LINUS". In: Y Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, vol. 482 of *Lecture Notes in Artificial Intelligence* Springer-Verlag.

Matheus, CJ, 1989. "A constructive induction framework". In: *Proceedings of the 6th International Workshop on Machine Learning*, pp 474–475, Morgan Kaufmann.

Mehra, P, Rendell, LA and Wah, BW, 1989. "Principled constructive induction". In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence* pp 651–657, Morgan Kaufmann.

Michalski, RS and Chilauski, RL, 1980. "Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis", *Policy analysis and information systems* **4**.

Michalski, RS, Carbonell, JG and Mitchell, TM, 1983. *Machine Learning: an artificial intelligence approach*, Vol. 1, Morgan Kaufmann.

Michalski, RS, Carbonell, JG and Mitchell, TM, 1986. *Machine Learning: an artificial intelligence approach*, Vol. 2, Morgan Kaufmann.

Michalski, RS, Davis, JH, Bisht, VS and Sinclair, JB, 1985. "Plant/ds: an expert consulting system for the diagnosis of soy-bean diseases". In: L Steels and JA Campbell, editors, *Progress in Artificial Intelligence* Ellis Horwood.

Michalski, RS, 1975. "Variable-valued logic and its applications to pattern recognition and machine learning". In: D Rine, editor, *Multiple-Valued Logic and Computer Science* North-Holland.

Michalski, RS, 1983. "A theory and methodology of inductive learning". In: RS Michalski, JG Carbonell and TM Mitchell, editors, *Machine Learning: an artificial intelligence approach*, vol 1, Morgan Kaufmann.

Michalski, RS, 1987. "Clustering". In: S Shapiro, editor, *Encyclopedia of artificial intelligence* Wiley.

Morik, K, 1989. "Sloppy modeling". In: K Morik, editor, *Knowledge Representation and Organization in Machine Learning* vol. 347 of *Lecture Notes in Artificial Intelligence* Springer-Verlag.

Michalski, RS and Stepp, RE, 1983. "Learning from observation: conceptual clustering". In: RS Michalski, JG Carbonell and TM Mitchell, editors, *Machine Learning: an artificial intelligence approach*, vol 1, Tioga Publishing Co.

Mitchell, TM, 1977. "Version spaces: a candidate elimination approach to rule learning". In: *Proceedings of the 5th International Joint Conference on Artificial Intelligence* Morgan Kaufmann.

Mitchell, TM, 1982. "Generalization as search", *Artificial Intelligence* 18 203–226.

Mitchell, TM, Utgoff, PE and Banerji, R, 1983. "Learning by experimentation: acquiring and refining problem-solving heuristics". In: RS Michalski, JG Carbonnel and TM Mitchell, editors, *Machine Learning: an artificial intelligence approach* pp 163–190, Tioga Publishing Co.

Muggleton, S and Buntine, W, 1988. "Machine invention of first order predicates by inverting resolution". In: *Proceedings of the 5th International Conference on Machine Learning* pp 339–351, Morgan Kaufmann.

Nilsson, NJ, 1980. *Principles of Artificial Intelligence* Tioga Publishing Co.

Quinlan, JR, 1986. "Induction of decision trees" *Machine Learning* **1** 81–106.

Quinlan, JR, 1987. "Generating production rules from decision trees". In: *Proceedings of the 10th International Joint Conference on Artificial Intelligence* pp 304–307, Morgan Kaufmann.

Quinlan, JR, 1990, "Learning logical definition from relations" *Machine Learning* **5** 239–266.

Rendell, L, 1986. "A general framework for induction and a study of selective induction" *Machine Learning* **1** 177–226.

Sablon, G, Ade, H, De Raedt, L and Bruynooghe, M, 1992. "Some thoughts on inverse resolution". In: S Muggleton, editor, *Inductive Logic Programming* Academic Press.

Sammut, C and Banerji, R, 1986. "Learning concepts by asking questions", In: RS Michalski, JG Carbonell and TM Mitchell, editors, *Machine Learning: an artificial intelligence approach* vol. 2, pp 167–192, Morgan Kaufmann.

Shapiro, EY, 1981. "An algorithm that infers theories from facts". In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pp 446–452, Morgan Kaufmann.

Shapiro, EY, 1983. *Algorithmic Program Debugging* MIT Press.

Subramanian, D and Feigenbaum, J, 1988. "Factorization in experiment generation". In: *Proceedings of the 5th American Conference On Artificial Intelligence*, pp 518–522, Morgan Kaufmann.

Tcheng, D, Lamber, B, Lu, SC and Rendell, L, 1989. "Building robust learning systems by combining induction and optimization", In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence* pp 806–812, Morgan Kaufmann.

Valiant, L, 1984. "A theory of the learnable" *Communications of the ACM* 27 1134–1142.

Vere, SA, 1975. "Induction of concepts in the predicate calculus", In *Proceedings of the 4th International Joint Conference on Artificial Intelligence* pp 282–287, Morgan Kaufmann.

Wrobel, S, 1988. "Automatic representation adjustment in an observational discovery system". In: D Sleeman, editor, *Proceedings of the 3rd European Working Session on Learning* Pitman.

Wrobel, S, 1989. "Demand driven concept-formation". In: K Morik, editor, *Knowledge Representation and Organization in Machine Learning*, vol. 347 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag.