Knowledge level model of a configurable Learning System

Céline Rouveirol[1], Patrick Albert[2]

[1] Laboratoire de Recherche en Informatique, URA 410 du CNRS, Équipe Inférence et Apprentissage, Université Paris Sud, bât 490 F-91405 Orsay, France. Email : celine@lri.fr.

[2] ILOG, 2, avenue Galliéni, BP 85, 94253 Gentilly Cedex, France. Email : albert@ilog.fr

Abstract. This paper presents the knowledge level model of a configurable learning system that follows a Generate and Test strategy. The knowledge level model makes explicit the elementary functionalities of the learning tool (referred to as *learning operations*), the control of the learning primitives (referred to as *bias*), and the different implementations of the learning primitives. The proposed model is based upon the inference structure formalism of KADS and will be used as an interface when interacting with the user. This explicit representation of learning operations and related bias will make the experimentation of different configurations of the proposed learning tool easier for a knowledge engineer developing a Knowledge Based application.

1 Introduction

The growing complexity of the knowledge based applications makes it necessary to use some kind of (semi-)automated techniques during the knowledge acquisition phase. Furthermore, knowledge engineering may require the use of various Machine Learning systems (ML) of different types, accepting different concept description languages, using different learning techniques, or managing different natures and complexities of the example set.

The knowledge engineer may thus need to quickly experiment different variants of the learning algorithms. In this context, she/he is faced to the following loop :

1. select a ML tool suited to her/his task,
2. set the parameters that control the behavior of the tool,
3. run the tool, analyze the results, stop or goto step 2 (or 1.).

In many cases, running the previous loop may be a difficult and tedious process. Selecting the appropriate tool may take time, and finding the setting of parameters that fits well to the problems characteristics may also be long and difficult.

Systems that provide a family of learning algorithms in a unified environment, rather than a single one, seem to provide an effective solution to shorten the complexity and duration of the above loop.

A first solution has been proposed within the Machine Learning Toolbox Esprit Project [9]. This project has led to the implementation of a ToolBox gathering a set of Machine Learning within a unifying global framework. The framework defines a common language for inputs and outputs, and a "consultant", implemented as an expert system, that helps the user in selecting the algorithms suited to her/his specific needs.

Though this system has proved to be a valuable step, the level of integration of the different algorithms is weak . Its building blocks are very large (full fledged learning systems), and the support of the user is quite low, but the variety of learning systems is quite large, ranging from numerical techniques to knowledge intensive symbolic learning systems. This systems implements one view of the approach, that may be well suited to experimented users, knowing the properties of the different tools and willing to experiment a variety of techniques.

We are working on a slightly different approach that proposes more support to the user, while restricting the range of available learning techniques. Our work aims at implementing a machine learning tool providing access to a family of learning algorithms (Generate and Test) and allowing the user to configure graphically different variations of algorithms and control parameters.

The support of the interaction with the user relies upon a knowledge level model of the Generate and Test systems. The knowledge level model provides both the set of modeling formalisms useful for representing the family of algorithms and the set of graphical notations that may be used to interact with the user.

The range of accessible learning tools is smaller than the MLT but we expect to provide a better support and smaller grained adaptation capability because we address only one class of machine learning system. We also expect to provide a simpler and more efficient interaction with the user, because of the use of a knowledge level model of the systems. Choosing a specific learning model, will be made by selecting high-level learning operations, and setting appropriate control parameters.

Within this overall objective, the paper presents the first results of this work : the definition of a set of knowledge level formalisms used to the represent the possible configurations, and the knowledge level model of Generate and Test learning methods. We emphasize specifically the role of *bias*, that represent in the ML terminology the parameters used to control the behavior of the different methods.

The rest of the paper is organized as follows. We first present our approach, and the research results in machine learning and in knowledge acquisition that we use. We follow by an in depth description of a generic knowledge level model of a class of learning systems, insisting on the nature and role of certains parameters (called bias) for describing and controlling their behavior.

2 A configurable ML tool

Our objective is to provide a configurable ML tool supporting a class of ML algorithms and that a user could configure graphically.

The interest of such a tool is described below :

- adaptable : depending upon her/his goals, upon the nature and complexity of the examples, and upon the type of domain theory available, the user will be able to assemble a set of learning procedures suited to the problem's characteristics and to instantiate the definition of the control parameters. This will greatly simplify the experimentation of different learning algorithms.
- simple and homogeneous : the configuration of a given learning system will be made by assembling a set of predefined learning operations, and by instantiating bias related to the chosen components. The unified graphical notation representing the possible decompositions of each steps of the learning algorithm will make it possible to specify graphically a given configuration in an intuitive way.

Such an objective is ambitious, but will make possible a wider use of ML techniques. By selecting a family of learning systems (`G-&-T`) and embedding them into a unified framework based upon an explicit knowledge level representation of the learning models underlying the algorithms, we will provide an easier access and understanding of the machine learning techniques.

As the objects that we represent are learning tasks, we need to represent their decompositions and the way they may be assembled. We use a knowledge level representation of the learning models based upon an extension [2] of the CommonKADS notations [39].

Mainly two formalisms are used:

A *Task decomposition tree*graphically represents the possible decompositions of the learning procedures into subtasks, while a specific notation makes explicit the possible alternatives while decomposing a task.

The *Inference Structure* notation provides a simple mean for expressing the data-flow within the decomposition of a task. It also provides a notation for specifying the configuration parameters that have to be set during the configuration process.

2.1 What is to be represented?

The objects to be represented are the learning operations, their parameters and their connections.

These entities are the data used by the configurer. The output of the configurer represents the resulting learning system : a set of connected learning operations with values for the associated control parameters.

We need to make explicit the set of possible configurations that represent all the possible decompositions of the top-level goal. The entities are the following :

learning operations : these are the operations to be realized by the learning algorithms. These operations may be primitives, corresponding to a piece of code, or abstracts, decomposed into less abstract operations.

possible decompositions of a learning operation : these are the different methods for implementing an abstract operation.

data-flow between learning operations: this represents the connection between operations within a decomposition.

control parameters: these parameters, control the behavior of the learning operations.

constraints controlling the decompositions: these entities, represent the conditions that attached to a given decomposition.

Except the latter, all these entities and their relations are represented graphically.

The controlling of a decomposition may be done at different levels of complexity : either by selecting one specific assembly of tasks, or by setting the value of parameters that control the behavior of a given assembly. The elicitation of such control parameters, named *bias*, is an important issue in the machine learning community (see section 3.1 for a precise description).

This work is tightly connected to different trends in machine learning and in knowledge acquisition. We describe below how it is related to research about the influence of bias and about adaptive machine learning on the one hand, and to research about knowledge level modeling on the other hand.

3 Inputs from Machine Learning and Knowledge Acquisition

3.1 Inputs from Machine Learning

The work presented here stems from the research in the ML community about elicitation of bias. The first paper in ML that referred to *bias* was [22]. In this paper, bias represents any kind of information (except examples of the target concept) that allows to prefer a target concept definition over a set of candidate ones. Bias is thus some additional information, not always explicitly stated within the learning system, expressing what a "good" target definition should be. This definition of bias is very general and since then, many authors in ML developed their own view of bias (see among others, [47], [16], and more recently [12], [15]).

We adopt here a more restricted and operational definition of bias. Given a set of learning algorithms that share the same overall structure (behavior), bias is control knowledge that allows to define a learning strategy within that structure. They represent a set of parameters that, besides the learning algorithms, define some of the characteristics of the learning tool and task. We have focussed here on three categories of bias. The so-called *Language bias*, that defines the target concept language, has been recognized for long as a major factor that influences learning. Another well identified bias, that will be referred in the rest of the paper as *Validation bias* states the properties on the target concept to be learned. Another kind of bias, which was less studied up to now, the *Search bias*, defines the dynamics of the learning system, such as for example, the type of search used to cross the search space [35].

More generally, biases represent the hypotheses underlying machine learning tools. Though their importance is well acknowledged, they are often hard-wired in the coding of the tools and thus are most often static. The user of the tool usually does not have access to their definition.

Recent works have focussed on elicitation of the biases used in different ML systems and on the definition of generic algorithms allowing to describe in a unified framework ML algorithms that have close behaviours. The generic algorithm presented here elaborates on GENCOL [11] that was the first generic algorithm for empirical concept learning proposed in ILP. The *ontology of bias* we propose is a synthesis (oriented by the purpose of building a ML system configurer) of previous works on declarative bias. Another line of research In ML that will benefit to our work studies the effects of bias (mostly language bias up to now) on learning results and performances. More positive results in this field will allow us to define a set of "safe" configuration rules.

3.2 Inputs from Knowledge Acquisition

The knowledge acquisition field has gained increased interest since the last ten years. It aims at proposing formalisms to represent knowledge based applications and methods to help the knowledge acquisition. This field is tightly connected to machine learning because of the similarity of the objectives, even if the methods differ.

Research in knowledge acquisition has produced a set of so-called Knowledge Acquisition Methods that provide both a set of graphical knowledge representation formalisms, and a set of directives used by the knowledge engineer to map the knowledge used by a human to solve a problem into a computer usable representation.

We can distinguish two main streams of results : the *problem specific methods* and the *problem independent methods*. The problem specific methods — as for example [18] [19] — are dedicated to a class of problems (diagnosis , configuration, etc.). They usually propose a knowledge representation dedicated to the problem, and a set of tools that operationalize the knowledge. Though very useful these methods lack of generality to satisfy our needs.

The "generic" approach, examplified by KADS [39], [50] Commet [42] or VITAL [30] aims at proposing general formalisms suited to describe knowledge level representations of problem solving methods representing how a problem, or a class of problems, may be represented and solved. These methodologies all come with the notion of library or of reusable components[6], [43]. We analyze below the adequacy of the two main generic knowledge acquisition methods to our needs.

Commet The Component of Expertise method (Commet) is mainly oriented toward the reuse of components. This method provides graphical formalisms for representing *task decomposition* and *control flow*. A task may be decomposed into subtask and the data-flow is graphically represented, allowing to specify the connection between subtasks and the domain entities onto which the tasks are applied.

Predefined problem solving methods may be used to precisely specify the type of decomposition used. Specific editors are used to define the data types of the parameters and a propagation engine ensures the coherency of the overall design.

An important characteristic of Commet is the dividing of the domain specific knowledge into two types of knowledge represented into two different models : the *domain model* that holds all of the domain dependent knowledge that does not change from one session to another, and the *case model* that holds all task specific knowledge what will be built and used during the use of the application.

This distinction is important regarding reusability, because the domain model may be more reusable than the case-model which is much more closely related to the problem solved by the specific application.

KREST [43], the tool associated to the methodology, puts the emphasis on reusability. It provides a service of library management used to store and retrieve predefined or user defined components. An important aspect of KREST is its ability to produce an executable program from the graphic specification.

CommonKADS CommonKADS shares some of the objectives of Commet but puts the emphasis at a different level. The main concern of CommonKADS is the reusability of knowledge level representations of classes of problems.

CommonKADS distinguishes three levels of modeling. The *domain level* that represents the domain dependent knowledge, the *inference level* that represents the problem dependent knowledge at an abstract level, using a extended data-flow formalism, and the *task level*, that represent the precise algorithmic of the task to be performed.

We are especially interested by the inference-level formalisms, the “Inference Structures”, that allow to specify both the problem solving components and their mapping onto the domain knowledge.

4 Selected formalism

Both approaches, KADS and KREST, provide elements, but none yet provides the solution, though recent advances in KREST may provide support for reasoning about tasks [48].

Among others, the two main functionalities that we need are not provided in the same framework :

- Commet provides a solution for representing the resulting configured system, and for generating the operational program from its specification ;
- KADS provides the formalisms to represent the set of components and their possible decompositions.

For the first step of our work, we have thus selected — and extended — the CommonKADS notations , for representing the knowledge level model of `G-&-T` systems. The models has been designed using KADS-TOOL [2] a case tool that supports the CommonKADS method, extended for supporting the decomposition of inference steps.

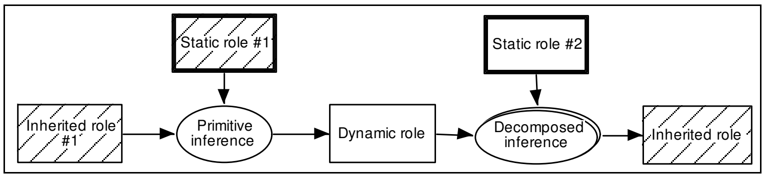

Fig. 1. Examples of graphical notations.

The inference structure shown in figure 1 contains the set of notations we rely upon. These notations, based upon CommonKADS, allow to represent both the possible configurations, and the resulting system. An *inference structure* shows the organization of a set of operations, the *inference steps*, represented by ellipsis. A simple ellipsis represents a primitive inference step, directly associated to an algorithm. A greyed ellipsis represents an intermediate inference step, that may be decomposed. Such an inference step may be decomposed through many inference structures. A simple box represents a *dynamic role*. It specifies the data flow between two or more inference steps. A bold box represents a *static role*, that is, a role that is used but not modified during the reasoning process.

The mapping between our entities and the CommonKADS concepts is as follows:

learning operations: these operations are represented by *inference steps*.

possible decompositions of a learning operation: these are the different *inference structures* that represent the decompositions of a composite inference step.

data-flow between learning operations: this is represented by the *dynamic roles* that link together the inference steps within an inference structure.

bias: the bias are represented by the *static roles* associated to the inference steps.

constraints controlling the decomposition: these entities, are not represented graphically.

5 Modeling Generate and Test Learning systems

5.1 Knowledge Level Model of Learning Systems

Let us first situate ourselves w.r.t. the idea of Knowledge Level Modeling of learning systems, especially with the work of Dietterich [13]. Our goal here is not to model a ML system integrated in a KA loop[3] or Knowledge Base system that improves its performance thanks to its learning component, as in [13]. Our focus here is modeling of ML systems *in the aim of building a configurable learning tool.* We are thus aiming at modeling learning systems in isolation of any performance system that may use their results. Moreover, recent research in ML (specially about bias in learning) seem to attenuate the conjecture of [13] about bias, or at least give a more respectable status to bias within learning systems, which may allow us to believe that it is possible to build conceptual models of ML system.

More practically, our claim is that complex learning systems, just like any complex apparatus or complex computer program should provide to their users some manual about the way they perform and consequently, some way to customize them to meet their actual needs. The adequacy of the Knowledge Level for this purpose, as well as for comparison of learning methods and elicitation of implicit (control) knowledge has already been discussed in [49].

This goal is very ambitious, we therefore address here a limited class of learning systems, and we do not provide for a complete guide of biases for such learning systems. The proposed model is fine-grained, sometimes as fine grained as one may object it does not belong to a Knowledge Level description anymore. We partly alleviate this objection by stating that for most biases that are considered to belong to the symbol level, a knowledge level characterization can be provided to/by the user as a first approximation, instantiations of such biases being delayed to the operationalisation phase, if any.

5.2 Generate-and-Test ML systems

The ML systems that are the scope of this paper are empirical learners. Unformally, one can state that they learn the definition of a target concept (denoted TC) from examples of TC (dynamic role *Environment* in the top level Inference Structure in fig. 2) and some background knowledge (dynamic role *Domain Theory*) which represents the domain model of the application domain. The most commonly used formalization in the Inductive Logic Programming community is : given a set of positive examples of TC, denoted E^+, a set of negative examples of TC, denoted E^- and some background knowledge, denoted BK, the learning goal is to find a hypothesis H such that $BK, H \models E^+$ and $B, H, E^- \not\models \Box$), where $\models$ denotes logical entailment and $\Box$ denotes the empty clause whihc means contradiction[4]).

[3] This is indeed another viewpoint when considering possible interactions between Knowledge Acquisition and Machine Learning, but out of the scope of this work.

[4] For a survey of different logical formalizations of empirical learning, see [26]

We focus here on empirical ML systems that use a *Generate and Test* strategy (denoted `G-&-T` in the rest of the paper) according to Mitchell's definition [21], as opposed to *Example Driven* learners.

Example driven learners modify their hypothesis space to dynamically adapt to newly incoming examples. More precisely, the learning process of example driven learners reduces HS by rejecting hypotheses of HS that do meet a validity criterion w.r.t. to a set of examples of TC. When the learning process is incremental, it is thus implicitly controlled by the order in which examples are provided or handled by the learning system. On the opposite, `G-&-T` learning systems, given a current hypothesis, first compute some relevant alterations of this hypothesis w.r.t. some strategic knowledge which is stated independently from the examples. The so generated hypotheses are then tested against some examples that validate or invalidate the proposed alterations. T

The `Generate-&-Test`[5] family of ML systems includes for example "historic" systems such as ID3 [32], CN2 [20], more recent ones such as MARVIN [37], INBF [41], ML-SMART [3], and Inductive Logic Programming systems (among all, FOIL [33], CLINT [10]). Note that these includes incremental systems that have a special policy w.r.t. the order of the examples as well as "batch" learning systems that handle a whole base of preclassified examples.

5.3 The KL model

In order to describe `G-&-T` systems at the knowledge level, we adopt a problem solving view of ML, that was first introduced in [21]: learning can be seen as a search for an appropriate definition of TC into a set of candidate definitions for TC. In the rest of the paper, a (partial) candidate definition for TC is called *hypothesis*, and the search space for a learning algorithm is denoted HS.

The learning task is decomposed into two tasks : the *computation of an initial representation* for HS and the *search* into HS.

The computation of the initial HS is mainly controlled by the so called *language bias* (section 6.2) which defines the language of the target concept definition, and a *search bias* that defines the direction of the search through HS.

The Search process in HS is controlled in the proposed model by two biases : a *search bias* (section 6.3), setting search strategies within HS and a *validation bias* (section 6.4) that specifies a validity criterion for hypotheses of HS. The learning process loops on those two tasks until the stopping criterion for learning, which is the validation of the final definition for TC (section 6.4), is fulfilled or until learning fails.

Those two tasks and their associated biases are interdependent. The chosen representation for HS will determine the type of search that will be performed and therefore the search bias. Another issue is about the relative importance of those two tasks w.r.t. to the overall learning process. On the one hand, when much information about the form of the target concept to be learned is available, it is computationnally interesting to put much attention to the task of computing

[5] See [29] for details about the ML aspects of the model presented here

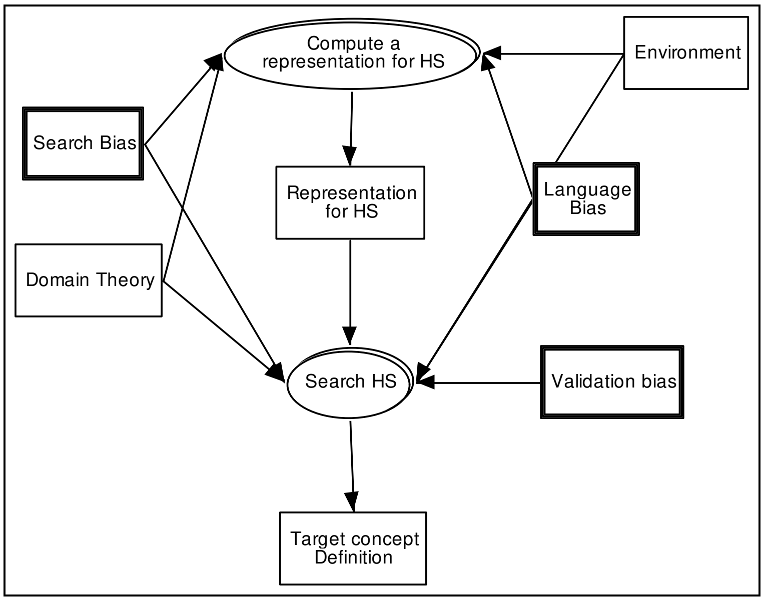

Fig. 2. The Problem Solving view of empirical learning

a representation of HS that "precompiles" the search process [4]. In this case, defining an adequate language bias alleviates most of the search problems. On the other hand, when few information is known beforehand about the target concept, the task of computing a representation for HS may be trivial. Therefore, most of the learning effort is left to the search process that, by combining an efficient search strategy and a clever access to the information stored in the examples of the target concept will allow to reach the expected TC definition.

6 Decomposition of bias

We now introduce a decomposition of biases for `G-&-T` algorithms (fig. 3). All of those biases, except for the search bias are applicable to other learning systems, including Example Driven ML systems. The role of each bias is illustrated by commenting the inference structures associated to each learning step. For all biases, a list of typical instantiations in `G-&-T` systems is given as well as some unformal rules about how they can be tuned to meet user's goals.

6.1 Representation of HS

We will assume in the remainder of the paper that HS is represented by a set of hypotheses that are a bound of HS : either the set of most specific or the set

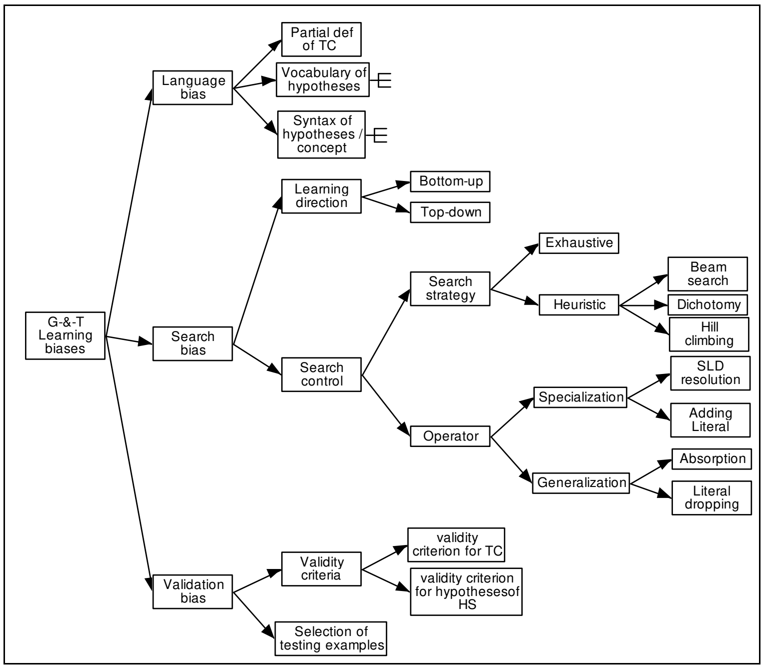

Fig. 3. Composition of bias for G-&-Tsystems

of most general hypotheses of HS that satisfy the validity criterion. This bound will be denoted CBD (for Current Best Definition). This representation requires that a generality relationship (a partial order) is explicitly given on the formulas of the target concept description language.

6.2 Language bias

The language bias defines restrictions on the language in which the target concept is expressed[6]. We give in the following a list of language biases commonly used in Inductive Logic Programming systems that learn target concept definitions in subsets of first order logic (see [45] and [4] for a more exhaustive survey).

Syntax and vocabulary of hypotheses of *HS*

– any constraint on the type/number of connectors, on the type/number of variables allowed in a hypothesis (namely, existentially and/or universally quantified variables) [10]. Sophisticated constraints on possible instantiations

[6] We do not consider as an explicit bias the language in which the examples and the background knowledge are expressed, although they have an influence on the target concept language.

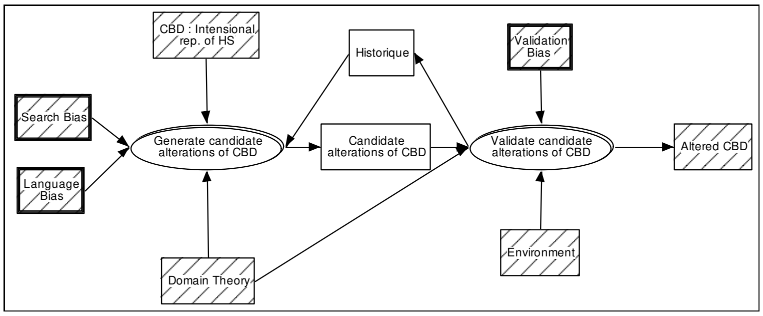

Fig. 4. Computation of alterations for the *Current Best Definition*

of such variables (determination, input/output modes, [38], ij-determination [27]) are also used.
- Higher order schemata [24], [45].
- For top-down systems such as [38], refinement graph that explicitly defines HS.

Syntax of the target concept The target concept language may be conjunctive; in this case, the final TC definition has to contain a single hypothesis, but still CBD contain several hypotheses that represent alternative choices for TC that should be resolved when learning stops. If the target concept language is disjunctive, the final TC definition is a set of hypotheses, such that each hypothesis is a partial definition of the target concept.

Influence of language bias The language bias acts *statically* when defining the initial representation for HS and also *dynamically* in the search process (see fig. 4). During search, it may be used to check that the alterations of CBD generated by the learning operators indeed belong to HS. Its influence on the learning results is crucial: the definition for TC must belong to the target concept language so that learning has any chance to succeed. With respect to the performance of the learning system, the size of HS and the complexity of the search through HS increases while the language bias weakens.

The exact influence of the syntax of hypotheses of HS on the performance of a ML system is still an open research issue in the ML community. Moreover, acquiring on even tuning such a bias may be quite difficult a “naive” ML system user. The two above objections raise the question of having such a bias as part of a knowledge level model of ML systems. Operationally, most of them are indeed justified by the necessity of bounding the complexity of the search process or by implementation choices that are not accessible at the level of detail of the inference structure. But, one may hope that syntactic criteria have some interpretation at the knowledge level: for example, finding a syntactically simple definition for the target concept (which translates into various syntactic criteria depending on the language bias) is something which, according to us, belongs to a Knowledge Level Model.

Configuration strategies Working in a conjunctive target concept language is the more constrained choice : learning will succeed if and only if at least one hypothesis of HS satisfies the stopping criterion. The vocabulary of the target concept language has therefore to be very carefully chosen. Working in a disjunctive concept language allows to express more complex concepts. Therefore, the choice of the vocabulary of TC has less influence on the success of learning but it also makes the search process in HS much more expensive. (see section 6.3). Therefore, the usual way to proceed is to start with a very strong syntactic bias on the hypothesis and/or concept language and to gradually shift to a more complex language when learning fails (see [1] or [10] for example).

6.3 Search bias

The search process, when the learning system handles a representation of HS with one of its bounds CBD, amounts to building a candidate alterations of CBD by applying a learning operator to it (fig. 5), and then checking the validity of the generated alterations against a set of examples for the target concept (fig. 6).

As an explicit generality relation is available to such learning systems, the search process has an interesting side effect on HS. For example, when the learning direction is bottom up and when a given CBD is not validated against examples of TC, all hypotheses more general than CBD are incorrect and must not be considered during the learning process (this is one reason why the dynamic role *Historic* appears in fig. 6). Note that we only consider in this paper search processes that monotonically reduces HS, therefore the alteration of CBD will be always from specific to general or from general to specific.

Learning direction CBD is modified by the application of a learning operator, one of a list of generalization or specialization operator. The user, depending on its learning goal will choose either to search HS from specific hypotheses to more general ones, CBD will therefore be a lower bound for HS, or to search HS from general to more specific hypotheses, in the latter case, CBD will be the upper bound of HS.

The choice of the learning direction is influenced by the validity criterion (see section 6.4) for the final TC definition. A top down learning algorithm is preferred when learning (maximally) discriminant concept definitions, that is, when the ML system user privileges learning complete concept definitions over learning correct concept definition. symmetrically, using a bottom up learning algorithm provides as a result more specific concept definitions that a top down algorithm does. Bottom up learning is more conservative and better suited when the correctness criterion is considered more important than the completeness one

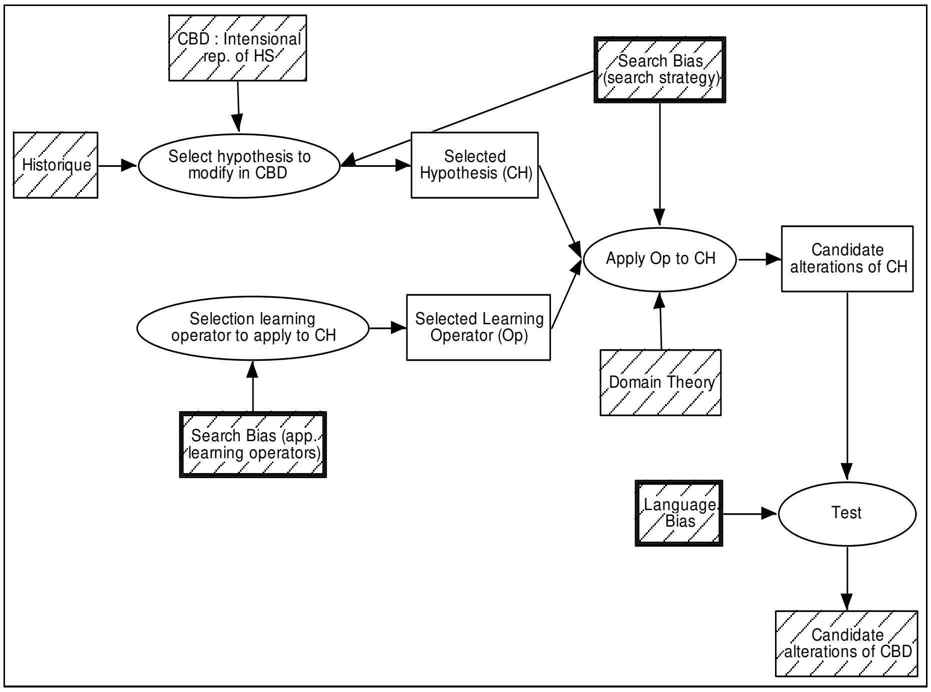

Fig. 5. Generation of candidate alterations of CBD

(that is, when it is not necessary to cover all positive examples but it is crucial that no negative example is covered by the learned concept definition).

The other search biases in `G-&-T` learning systems that operate on a representation of HS by a bound define dynamic constraints on possible alterations of CBD. They can be decomposed into :

Applicable learning operators Several learning operators have been described in ML. We have extracted from the Inductive Logic Programming literature the following (non exhaustive) list :

1. the dropping literal generalization operator that drops a literal in the body of a clause ([10], [34]), and its dual operator for specialization that adds literals to the body of a clause [3], the well known *least general generalization* operator ([31], [27]).
2. for operators that dynamically make use of the domain theory : the inverse resolution operator [25] and its dual operator for specialization, resolution [3].

Each learning operator generates its own search space denoted H_{Op}, which may not coincide exactly with HS. An operator may not be complete with

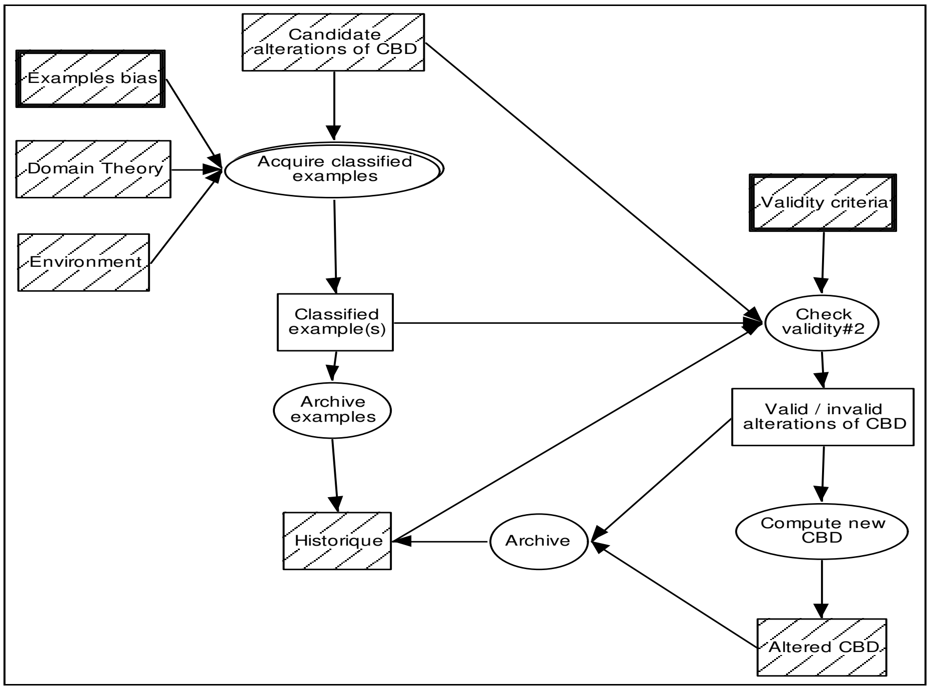

Fig. 6. Validity checking of candidate alterations of CBD

respect to the language bias, i.e., they may not be able to reach all hypotheses of HS. Thus, chosing an operator which is incomplete w.r.t. the language bias is a way to prune hypotheses of HS that will never be considered through learning (the search bias is then restrictive, in the terminology introduced by [36]).

Search strategy In case the target concept language is disjunctive (CBD may therefore contain more than one hypothesis), the search strategy defines an order on hypotheses of CBD that are to be altered by the learning operator. The set of possible strategies here are the usual ones in Problem Solving : depth first, breadth first, iterative deepening for uniformed strategies, best first, hill climbing, stochastic [23], etc. If the target concept is conjunctive, no choice is to be done at that step.

Any strategy can be defined to search H_{Op}. One may choose to exhaustively search through H_{Op} (smallest step [28], [10]) which is a careful strategy, considering that each generated alteration has to be validated. This strategy is applied, when the user does not want to degrade theoretical properties for the learning algorithm (see for example, [10], [14], [8]) are defined with the hypothesis of exhaustive search), or when the domain theory is incorrect or incomplete. The largest possible step w.r.t. to given operator allows (as in [41]) to minimize memory storage. Dichotomy search (as described in [17]) minimizes the number of examples needed for learning.

Each of these strategies is adapted to specific learning situations (expressed in terms, for example of distributions of the positive and negative examples of TC in the instance space, the space of all possible example descriptions) and learning goals (such as minimizing the number of intermediate CBD to build before learning, bounding the estimation of the error rate of the target concept definition learned, minimizing memory storage, etc.)

6.4 Validation bias

The biases of validation include:

Validity criterion that assess the validity of an intermediate CBD. Note that this criterion may be different than the stopping criterion that assesses the final target concept definition to be learned. For example, when the target concept language is disjunctive a partial definition is validated when it is correct but not necessarily complete, whereas the final TC definition has to be correct **and** complete. The most used validity criteria are the *completeness* criterion : a hypothesis of HS has to cover all positive examples of TC and the *correctness* criterion : a hypothesis of HS must not cover any negative example of TC. These are absolute validity criteria, usually applied in incremental learning systems. When a pre-classified database of examples for the target concept is available, a heuristic measure evaluating some gain in discrimination or some expected error rate of the learned target concept can be applied (for example, [32], [33], [3]).

Selection of testing examples. A candidate alteration of CBD which is acceptable w.r.t. the language bias has to be checked against the above validity criteria against a set of examples. The subset of examples of TC that are pertinent for validating the alteration of a given CBD into a new bound $NCBD$ (denoted S_{pex}) is defined as: for a bottom up strategy, the set of examples of TC covered by $NCBD$ and not by CBD ($NCBD$ is more general than CBD) and and symmetrically for a top down strategy the set of examples of TC covered by CBD and not by $NCBD$ ($NCBD$ is then more specific than CBD). Still, this set of examples may be large, and when the validation mode is a set of queries to the user, enumerating all such examples may be tedious. Diverse heuristics can then be applied to reduce this set, sometimes up to a singleton (see for example [5]), increasing the risk to miss an example that would contradict the alteration.

7 Related work

Our work takes place at the border between machine learning and knowledge acquisition. One promising and related approach, illustrated by [46], is to use

KADS domain level of a given application to control machine learning. In this work, the model driven analysis promoted by KADS defines *what* has to be learned, and its role within the overall knowledge based application.

Some works in the Inductive Logic Programming are also close to ours : GENCOL [11] (as quoted in section 3.1, MILES [44] which also has some strong links with our work, but up to now, has put much more stress on the study of language bias. Our originality w.r.t. those two works is our representation in a Common KADS formalism of our model on the one hand, and our explicit concern of building a configurer that will use our knowledge level model for interaction with the user on the other hand.

Within the knowledge acquisition community, [49] argues on some possible roles of Machine Learning within a Common Kads Methodology. We agree on the fact ([49], p13) that the Knowledge Level is a good level of description for analyzing, comparing and here configuring learning systems. Some very recent work, presented by [40], provides a configurable library of a family of ID3 like learning algorithms within KREST, is exactly in the same philosophy than ours.

Finally, let us quote a work [7] about the status and representation of control in a knowledge in KA methodologies that is quite related to our concern of building a knowledge level model of `G-&-T`systems parameterisable in terms of bias.

8 Conclusion - Perspectives

We have presented first results for building a configurable machine learning kit. The basis of the configurer is a knowledge level description of a family of empirical learners represented as a set of decomposed inference structures. This formalism provides a knowledge level representation of learning models and supports the explicit representation of some bias. The ML system is assembled by choosing decompositions of learning operations and by setting the associated biases.

As a first step, we have used these formalisms to represent the `G-&-T` family of learning algorithms. This work, though still incomplete, seems to validate the approach. As a useful side-effect, the knowledge level modeling puts the emphasis on bias and their role in the learning process. The models have been designed with KADS-TOOL and a first mockup has been implemented in Prolog.

The next step is now to complete the model, to formalize the configurer (and especially answering the questions related to the dynamic configuration process), and to set up a first prototype allowing some experiments.

Acknowledgements

This work is partially supported by ESPRIT III BRA 6020 (Inductive Logic Programming) and by the French MRT through PRC-IA. The authors thank Claire Nédellec for her involvement in the definition of the generic model which is the basis of this paper and Karine Causse for most helpful comments on the

KA aspects of this paper. This work has also benefited from discussions with members of the ILP project, specially those participating in the Declarative Bias Task. We would like to finally thank the anonymous referees of this paper for very useful suggestions.

References

1. H. Ade, L. De Raedt, and Bruynooghe M. Declarative bias for specific-to-general ilp systems. *Machine Learning*, 1994. Special Issue on Declarative Bias, submitted.
2. P. Albert, E. Brunet, J. Gonzague, and I. Bousquet. Kads-tool, a case tool for the *common*kads methodology. In *Conference of International Association of Knowledge Engineers*, 1992.
3. F. Bergadano and A. Giordana. A knowledge intensive approach to concept induction. In *Proceedings of the 5th International conference on Machine Learning*. Morgan Kaufmann, 1988.
4. F. Bergadano and D. Gunetti. Learning clauses by tracing derivations. Technical report, University of Torino, 1993.
5. R.C. Berwick. Learning from positive-only examples : The subset principles and three cases studies. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning II : An Artificial Intelligence Approach*, pages 625–646. Morgan Kaufmann, 1986.
6. Breuker, Bredeweg, Valente, and Van de Velde. Reusable problem solving components: the *common*kads library. In *3rd KADS User Meeting, Munich.*, 1993.
7. K. Causse. A model for control knowledge. In *Proceedings of the 8th Knowledge Acquisition for Knowledge-Based Systems Workshop (Banff'94)*, Banff, Alberta, Canada, 1994.
8. W. Cohen. Learnability of restricted logic programs. In *Proceedings of the Inductive Logic Programming Workshop*, pages 41–71, 1993. Bled, Slovenia.
9. MLT CONSORTIUM. Final public report. Technical report, 1993. Esprit II Project 2154.
10. L. De Raedt. *Interactive Theory Revision: an inductive logic programming approach*. Academic Press, 1992.
11. L. De Raedt and M. Bruynooghe. A unifying framework for concept-learning algorithms. *The Knowledge Engineering Review*, 7(3):251–269, 1992.
12. M. Desjardins. *Proceedings of the AAAI 92 workshop on Declarative Biases*. 1992.
13. T.G. Dietterich. Learning at the knowledge level. *Machine Learning*, 1:287–316, 1986.
14. S. Džeroski, S. Muggleton, and S. Russel. Pac learnability of logic programs. In ACM, editor, *Proceedings of the 1992 Workshop on Computational Learning Theory*, pages 128– 135, 1992.
15. D. Gordon. *Proceedings of the ML92 workshop on Biases in Inductive Learning*. 1992. Aberdeen.
16. B. Grosof and S. Russell. Shift of bias as non-monotonic reasoning. In P.B. Brazdil and K. Konolige, editors, *Machine Learning, Meta-Reasoning and Logics*. Kluwer Academic Publishers, 1990.
17. D. Haussler. Applying valiant's learning framework to ai concept-learning problems. In Y. Kodratoff and R.S. Michalski, editors, *Machine Learning : an artificial intelligence approach*, volume 3. Morgan Kaufmann, 1990.

18. S. Marcus. Salt: a knowledge acquisition tool for propose and revise systems. In Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic, 1988.
19. S. Marcus and J. McDermott. Salt: A knowledge acquisition language for propose and revise systems. *Artificial Intelligence*, 39(1):1–38, 1989.
20. R.S. Michalski. A theory and methodology of inductive learning. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning : an artificial intelligence approach*, volume 1. Morgan Kaufmann, 1983.
21. T.M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
22. T.M. Mitchell. The need for biases in learning generalizations. In *Readings in Machine Learning*. Morgan Kaufmann, 1991.
23. D. Mladenic. Combinatorial optimization in inductive concept learning. In *Proceedings of Int. Conf. on Machine Learning*. Morgan Kaufmann, 1993.
24. K. Morik, S. Wrobel, J.-U. Kietz, and W. Emde. *Knowledge Acquisition and Machine Learning - Theory, Methods, and Applications*. Academic Press, 1993.
25. S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th Int. Conf. on Machine Learning*, pages 339–351. Morgan Kaufmann, 1988.
26. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*. to appear.
27. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st conference on algorithmic learning theory*. Ohmsha, Tokyo, Japan, 1990.
28. C. Nédellec. How to specialize by theory refinement. In *Proceedings of ECAI-92*, 1992.
29. C. Nédellec and C. Rouveirol. Hypothesis selection biases for incremental learning. In *Proceedings of the AAAI Spring Symposium Series workshop on Issues for Incremental Learning*, 1993.
30. L. O'Hara. Knowledge acquisition in vital. Technical Report D2.1.2, Esprit Project 5365, 1992.
31. G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5. Edinburgh University Press, 1970.
32. J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1983.
33. J.R. Quinlan. Learning logical definition from relations. *Machine Learning*, 5:239–266, 1990.
34. C. Rouveirol. Extensions of inversion of resolution applied to theory completion. In S. Muggleton, editor, *Inductive Logic Programming*, pages 63–93. Academic Press, 1992.
35. C. Rouveirol. *Working Notes of the Declarative Bias MLNet workshop*. 1994. Catania.
36. S. Russell and B. Grosof. A sketch of autonomous learning using declarative bias. In P.B. Brazdil and K. Konolige, editors, *Machine Learning, Meta-Reasoning and Logics*, pages 19–54. Kluwer Academic Publishers, 1990.
37. C. Sammut and R. Banerji. Learning concepts by asking questions. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning : an artificial intelligence approach*, volume 2, pages 167–192. Morgan Kaufmann, 1986.
38. Ehud Y. Shapiro. *Algorithmic Program Debugging*. The MIT press, 1983.
39. G. Shreiber, B. Wielinga, and J. Breuker. *KADS: a Principled Approach to Knowledge-Based System Developpement*. Academic Press, 1992.
40. A. Slodzian. Configuring decision tree learning algorithms with krest. Technical report, VUB, 1994.

41. B.D. Smith and P.S. Rosembloom. Incremental non-backtracking focusing: A polynomially bounded generalization algorithm for version space. In *AAAI-90 : proceedings of the ninth conference*, pages 848–853. Morgan Kaufmann, 1990.
42. L. Steels. Reusability and configuration of applications by non-programmers. Technical Report AI-memo 92-4, AI Lab. Free University of Bruxelles, 1992.
43. L. Steels. The componential framework and its role in reusability. In David J-M., Krivine J-P., and Simmons R., editors, *Second Generation Expert Systems*. Springer-Verlag, 1993.
44. B. Tausend. Biases and their effects in inductive logic programming. In Springer Verlag, editor, *Machine Learning: European Conference on Machine Learning-94*, pages 431–434, 1994.
45. B. Tausend. Representing biases for inductive logic programming. In Springer Verlag, editor, *Machine Learning: European Conference on Machine Learning-94*, pages 427–430, 1994.
46. J. Thomas, J.G. Ganascia, and P. Laublet. Model-driven knowledge acquisition and knowledge-biased machine learning. In G. Tecuci, S. Kedar, and Y. Kodratoff, editors, *Proceedings of the IJCAI-93 Workshop on Machine Learning and Knowledge Acquisition*, pages 220–235, 1993.
47. P.E. Utgoff. Shift of bias for inductive concept-learning. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning : an artificial intelligence approach*. Morgan Kaufmann, 1986.
48. W. Van de Velde. Invited talk: Toward a toolkit for knowledge engineering. *Journées Acquisition des Connaissances*, 1994.
49. W. Van de Velde and A. Aamodt. Machine learning issues in commonkads, 1992.
50. B. Wielinga, W. Van de Velde, G. Shreiber, and D. Akkermans. The *common*kads framework for knowledge modelling. In *AAAI Knowledge Acquisition Workshop 1992, Banff, Canada*, 1992.

This article was processed using the LaTeX macro package with LLNCS style