Controlling the Complexity of Learning in Logic through Syntactic and Task-Oriented Models

Jörg-Uwe Kietz, Stefan Wrobel
GMD (German Natl. Research Center for Comp. Science)
Institute for Applied Information Technology
Pf. 1240, W-5205 St. Augustin 1, Germany
kietz@gmdzi.gmd.de, wrobel@gmdzi.gmd.de

Abstract

Due to the inadequacy of attribute-only representations for many learning problems, there is now a renewed interest in algorithms employing first-order logic or restricted variants thereof as their knowledge representation. In this paper, we give a brief overview of the dimensions along which the complexity of learning in such representations can be controlled. We then present *RDT*, a model-based learning algorithm for function-free Horn clauses with negation that introduces two new means of complexity control, namely the use of syntactic rule models, and the use of a task-oriented domain topology. We briefly describe some preliminary application results of RDT within the knowledge acquisition system MOBAL, and present directions of further research.

1 Introduction

As described by Mitchell [Mit82], concept learning from examples can be seen as a process of search in a space of hypotheses that is ordered by the generality relationship between hypotheses. The size of this space and the properties of its generalization relation determine the difficulty of finding the desired target concept, i.e., a concept that covers all positive examples (completeness) and none of the negative examples (consistency). For attribute-based (propositional, feature-vector) formalisms, the space of hypotheses is finite, and there are only a few generalizations/specializations of any given hypothesis. As a result, it has been possible to construct very efficient learning programs for attribute-only languages, such as ID3 [Qui83] and its successors.

In many practical applications, however, attribute-based representations are insufficient. This includes domains where structural properties or time-dependent relationships need to be expressed, as in medical diagnosis [Wro90][1]. In response, learning in representations that are variants of first-order logic (FOL) has received more interest recently. Here, the learning problem can generally be stated as follows. Given a background theory B, and sets of positive and negative examples E^+ and E^- (ground instances of the concept to be learned), find an inductive hypothesis H such that

$$B \cup H \vdash E^+, \text{but } B \cup H \not\vdash E^-$$

In the case of full first-order logic, the hypothesis space is of course infinite, and worse, the generalization relation has very bad properties. Since logical inference in the general case is undecidable, it is even undecidable whether a hypothesis covers an example, or whether one hypothesis is a generalization of another.

To deal with such complex hypothesis spaces, the designer of a learning algorithm for a logic-based formalism can make choices along a number of dimensions:[2]

- Choice of representation formalism
- Restricted search subspaces
- Restricted background knowledge
- Search control

In this paper, we present an algorithm called RDT[3], an approach to learning in logic that we want to characterize along the above dimensions. Before discussing RDT, however, it will be useful to elaborate on what these dimensions are, and which choices other algorithms have made (section 2). This will make it easier to see where RDT uses proven techniques, and where it contributes novel approaches. In section 3, we will describe RDT's choices on the first three dimensions, i.e., its knowledge representation and learning model. Section 4 is entirely devoted to the fourth point, search control, and contains a detailed description of the RDT algorithm. In section 5, RDT is placed in the context of the knowledge acquisition system MOBAL of which it is a part, and some preliminary application results of RDT/MOBAL are described. Finally, we conclude with a discussion of further research in section 6.

2 Dimensions of Controlling Complexity

Choice of representation formalism An immediate way of reducing the complexity of learning logical expressions is to use subsets of FOL. To our knowledge, full FOL is not used by any existing learning algorithm except Plotkin's relative

[1] See [Qui90] for a detailed discussion of the limits of attribute-only representations.
[2] We are ignoring the use of *oracles*, which is also a common approach.
[3] RDT stands for Rule Discovery Tool.

least general generalization (RLLG) method [Plo70], which uses definite clauses, but was severely limited in its implementation [Mug90]. Instead, most research concentrates on Horn clause logic (pure Prolog) and subclasses thereof, in particular the subclass of function-free Horn clause programs (Datalog programs, [CGT90]) in which the generalization relationship is decidable. This choice of representation was made eg. in ARCH [Win75], INDUCE [Mic83], MARVIN [SB86], CLINT [DB89], and FOIL [Qui90]. MIS [Sha83], even though operational in full Horn clause logic, suffered from complexity problems that prevented it from learning certain concepts even though they were in its hypothesis space [Mug90]. Other programs that use full Horn clauses as their basic hypothesis space typically use other restrictions to reduce the size of the space (eg. CIGOL, IRES, LFP2, or GOLEM, see below).

Restricted search subspaces Once the base representation has been chosen, further restrictions are often made to determine a hypothesis subspace that will actually be searched. This can be done either by explicitly defining language subsets through restrictions on the form of hypotheses, or by using generalization/specialization operators that produce only selected members of the hypothesis space. For example, DeRaedt's CLINT [DB89] uses a series of languages that are defined by allowing increasingly more existential variables (variables not found in the head of the clause), and increasingly more relations between those variables. In GOLEM [MF90], the related notion of *ij-determinacy* is used that limits the indeterminate introduction of new variables in the body of a clause. Both of those restrictions are ways of defining subspaces of the general Horn clause hypothesis space.

The alternative way, i.e., using restricted operators, has been used eg. in the CIGOL system [MB88]. This system uses inverse resolution generalization operators (absorption and identification) that are in principle capable of producing any Horn clause expression, but are restricted to unit clauses to reduce their complexity. Rouveirol and Puget [RP89] discuss the difficulties involved in removing this restriction for their IRES system. LPF2 [Wir89], also based on inverse resolution, is not restricted to unit-clauses, but uses restrictions on the inverse substitutions that may be assumed.

Restricted background knowledge In Horn clause logic, the generalization relationship between two clauses is easy to handle without background knowledge [Bun88]. Thus, the form of allowed background knowledge has an important influence on the complexity of the learning task. One often used restriction is the use of background knowledge only in the form of ground unit clauses (facts). In MARVIN [SB86], this is done by forward applying the available inference rules to generate additional facts about an example ("elaboration" of the example). In GOLEM [MF90], a program based on an extension of Plotkin's relative least general generalization (RLLG) approach, the background knowledge is required to be "generative", which again means that all derived unit clauses will be ground.

Search control Besides using the generalization relation to prune its search (which is guaranteed not to lose relevant hypotheses), a program can use heuristics to guide its search in the hypothesis space, i.e., it can improve efficiency at the expense of completeness — some concepts that are in its hypothesis space will not be found, but those that are found are found much quicker. Such an approach has been used in the AQ/INDUCE program [Mic83], a bottom-up (generalizing) learner where a user-settable criterion (LEF) was used to determine which generalizations to keep in the focus of the system's beam search. The CIGOL system [MB88], also a bottom-up learner, uses information compression as a measure to judge promising or less promising generalizations. In FOIL [Qui90], a top-down (specializing) learner, the information gain measure popular from ID3 is used to judge whether the addition of a literal improves the current hypothesis. This makes the system short-sighted, since it rejects the addition of literals the positive effect of which is seen only after more literals have been added (eg. literals that restrict existential variables).

The choices made in RDT

The RDT learning algorithm presented here follows some established paths through the complexity jungle:

- it uses function-free Horn clause logic as its base representation (see 3.1);
- it uses background knowledge by transforming it into a set of ground facts in a manner similar to MARVIN, RLLG or GOLEM (see 3.2);
- it is a top-down general-to-specific learner like FOIL or MIS that uses the generality ordering of the hypothesis space to prune its search (see 4.1).

In other respects, however, the program charts some new terrain unvisited by other approaches:

- RDT defines the hypothesis subspace that is actually searched not via generalization or specialization operators or abstract restrictions on hypotheses, but via a set of explicitly spelled out hypothesis templates (*rule models*, see 3.3);
- In addition to the syntactic restrictions, RDT introduces a way of further restricting the hypothesis subspace in a *task-oriented* fashion by allowing the user to specify a domain *topology* (see 3.4);
- On the strategy dimension, RDT uses an automatically constructed *sort lattice* to dynamically prune its search (see 4).

In the following sections, we will describe each of the above choices in more detail.

3 Representation and Learning Model

3.1 Definition of the representation

RDT uses the function-free subset of Horn-clause logic as its knowledge representation formalism[4], and it extends it by allowing negated literals. Following the terminology from [Llo87], we can define this representation more precisely as follows[5]. A *predicate* is denoted by a predicate symbol (usually p, q, r ...) and has a fixed arity a. A *term* is a constant (denoted by c_1, c_2, or other lowercase letters), or a variable (denoted by X, Y, or other uppercase letters). An *atom* is a predicate applied to terms, i.e., of the form $p(t_1, \ldots, t_a)$. A *positive literal* is an atom, a *negative literal* is the negation of an atom, i.e., $\neg p(t_1, \ldots, t_a)$. A *clause* is a finite disjunction of literals where all variables are implicitly universally quantified. It is usually written as an implication:

$$C \leftarrow L_1, \ldots, L_n$$

where C, the *conclusion*, and the L_i, the *premises*, are positive or negative literals.

$$C \vee \neg L_1 \vee \ldots \vee \neg L_n$$

is the corresponding *clausal form* of the notation.

- If $n = 0$, the clause is called a *fact*, and usually written without the $\leftarrow$. All facts are required to be *ground*, i.e., contain no variables.
- If $n \geq 1$, such a clause is called a *rule*. In contrast to Prolog and eg. FOIL [Qui90], negative literals as premises of a rule are not treated as negation by failure, but as proper negation, i.e., negative premises require explicitly negated facts in the knowledge base to be satisfied.

RDT's knowledge representation consists of facts and rules as described here.

3.2 Examples, background knowledge and concepts

The examples given to RDT are sets of positive and negative facts about a target predicate p that define the desired concept extensionally. For the target predicate "related_to", such facts might be

related_to(john,vivian)
related_to(peter,sandy)
¬related_to(mary,mike)
...

[4]The MOBAL system, of which RDT is a part, uses a more complex representation. Here, we describe only the subset used by RDT.

[5]In this paper, RDT's knowledge representation is described using the standard logical notation to improve readability; the system actually uses a syntactically different format.

Background knowledge about the examples can be provided in the form of additional facts and rules, eg.

husband(john,vivian)
father(peter,sandy)
¬married(peter,janice)
unrelated(mary,mike)
age(mary,27)
married(X,Y) ← husband(X,Y)
married(X,Y) ← wife(X,Y)
...

All background knowledge rules are used to forward-infer additional facts from those that were input[6], so that for the purpose of learning, all background knowledge effectively consists of facts. Here, the system would derive only

married(john,vivian)

To ensure that these facts are ground, only those rules are used for inferencing where all conclusion variables occur in the premises also ("generative" rules in the terminology of [MF90]) . A depth-limit is used to keep the expense of this forward inferencing process under control. Within the depth limit, the system effectively generates a ground model of its background knowledge. This is a technique that is also used in MARVIN [SB86] (the "elaboration" of the examples), and in GOLEM [MF90] (the generation of h-easy ground Herbrand models for generalization).

The goal of RDT is to find a definition for the target concept[7] that consists of a set of rules, eg.

related_to(X,Y) ← mother(X,Y).
related_to(X,Y) ← father(X,Y).
related_to(X,Y) ← related_to(X,Z), related_to(Z,Y)
related_to(X,Y) ← ¬unrelated(X,Y)
...

The rules are not required to cover all instances of the target concept, and they may overlap (see section 4). All learned rules are entered into the knowledge base, and thus become background knowledge for further learning (closed-loop learning [Emd89b]).

3.3 Rule models

In the preceding section, we have defined RDT's base representation that defines its maximal space of hypotheses. Now, we will present the means by which the subspace

[6] This task is performed by IM2 [Emd89a], the inference engine of the MOBAL system.
[7] Or the negation thereof, see section 5.

that will actually be searched is defined. The first such means, *rule models*, is of a syntactical nature. By a rule model (or rule schema)[8], we mean a rule in which predicate variables are used instead of actual domain predicates.

More precisely, a *predicate variable* (denoted by P, Q, or other uppercase letters) is a variable that can be instantiated only to predicate symbols of a certain arity a. An *atom schema* is a predicate variable applied to terms, i.e., of the form $P(t_1, \ldots, t_a)$. A *literal schema* is an atom schema or the negation of an atom schema. A *rule schema* (usually denoted R or RS) is a finite disjunction of literal schemas of length at least two, where all (non-predicate) variables are implicitly universally quantified. It is usually written as an implication ($n \geq 1$):

$$C \leftarrow L_1, \ldots, L_n$$

An *instantiation* Σ is a finite set of pairs P/p, where P is a predicate variable, and p is a predicate symbol, both of the same arity. The set of all p is called the *range* of Σ. The application of an instantiation Σ to a rule schema R (denoted $R\Sigma$) denotes the result of replacing each predicate variable mentioned in Σ by the corresponding predicate symbol. R is called *predicate ground* if all its predicate variables are instantiated, i.e., R is a rule.

Based on the set of rule models $\mathcal{R}$ provided to RDT, and the set of domain predicates $\mathcal{P}$ in the knowledge base, the hypothesis subspace that is searched is defined as the set

$$\mathcal{H} = \{R\Sigma | R \in \mathcal{R} \wedge range(\Sigma) \subseteq \mathcal{P} \wedge R\Sigma \text{ predicate ground}\}$$

i.e., as the set of all possible predicate ground instantiations of rule models with domain predicates.

This way of defining a hypothesis subspace can be contrasted with the use of specialization operators as in MIS [Sha83], or FOIL [Qui90]. Even though $\mathcal{R}$ is simply an unordered list of rule schemas, a generalization relation on $\mathcal{R}$ can be defined by suitably extending the standard clause subsumption relation. Before presenting our extension of subsumption for rule models (section 4.1), let us elaborate on how such a generalization relation can then be used. Given a hypothesis H and $R \in \mathcal{R}$ such that $H = R\Sigma$, we find all $R' \in \mathcal{R}$ that are specializations of R, and instantiate them in all possible ways (see section 4.1ff. for the details of this process). Thus, $\mathcal{R}$ constitutes an extensional definition of a specialization operator on hypotheses: it defines the operator by listing the form of hypotheses it produces, where normally the form of hypotheses is determined by the operator specification.

The advantage of this approach is that the hypothesis space is under very explicit user control. Based upon the elements of $\mathcal{R}$, it is easy to see for the user which

[8]In this paper, the terms rule model and rule schema are used interchangeably. The former emphasizes the functional aspect as a model of hypotheses, the latter the structural aspect of a rule with predicate variables.

form the learnable hypotheses will have[9]. Furthermore, the hypothesis space can be controlled in a very finely grained fashion, as individual hypotheses forms can be removed or added at will. By partially instantiating rule schemata, the user can even incorporate knowledge about individual domain predicates, as in eg.

Q(S1) ← P1(S1), preceding_state(S1,S0), P2(S0)

where the name of the state successor predicate is explicitly given to avoid having to test other binary predicates in its place. All of this would be more difficult if a general operator specification had to be changed. The disadvantage of this form of a learning model is the need to define the set $\mathcal{R}$ for an application, which is not always easy; often, a standard set of rule models is used. In section 6, we discuss ways of extending $\mathcal{R}$ automatically through the use of rule schema modification operators.

The approach presented here originated in Emde's METAXA system [Emd87] and was also used in the BLIP system [Wro88]. DeRaedt and Bruynooghe [dRB89] have examined the use of rule schemata as defined here to propose new concept definitions by analogy (the CIA/CLINT technique).

3.4 The topology

Almost all learning approaches cope with the complexity of learning in logic by defining various syntactic restrictions on their hypothesis space or learning operators. This ignores an important source of control for learning — the use to which the learning results are to be put. Any learning program is used to achieve a specific goal, its output is to subserve a specific task. Thus, task-oriented restrictions on the hypothesis space can tap into an important source of efficiency.

In RDT, we implemented such a task-oriented restriction in form of a predicate *topology* [Mor90]. A predicate topology consists of a grouping $\mathcal{T} = \{T_1, \ldots, T_m\}$ of the predicates $\mathcal{P}$ into possibly non-disjoint sets T_i, the so-called topology *nodes*, each of which is labelled with a mnemonic name by the user. The topology nodes, in turn, can be connected to each other by directed links to form a hierarchy. This hierarchy is used to further restrict the hypothesis space of RDT by allowing only those hypotheses that use predicates within a topology node, or that use predicates within one topology node in the premises, and a predicate from one of its parents in the conclusion. More precisely, the hypothesis space $\mathcal{H}^{\mathcal{T}}$ can be defined as follows:

$$\mathcal{H}^{\mathcal{T}} = \{H \in \mathcal{H} | H = p_{concl} \leftarrow P_{prems} : \exists T_i \in \mathcal{T} : p_{concl} \in T_i \wedge P_{prems} \subseteq T_i \cup children(T_i)\}$$

where $children(T_i)$ denotes the union of the direct subnodes of T_i.

The topology nodes thus provide a means of subdividing a domain into regions with different purposes, and the links allow the user to restrict the hypotheses that

[9]This can be compared to a requirement on "functional" knowledge representations that originated in the KL-ONE school [BS85]: that the set of inferences provided be easily recognizable by a user.

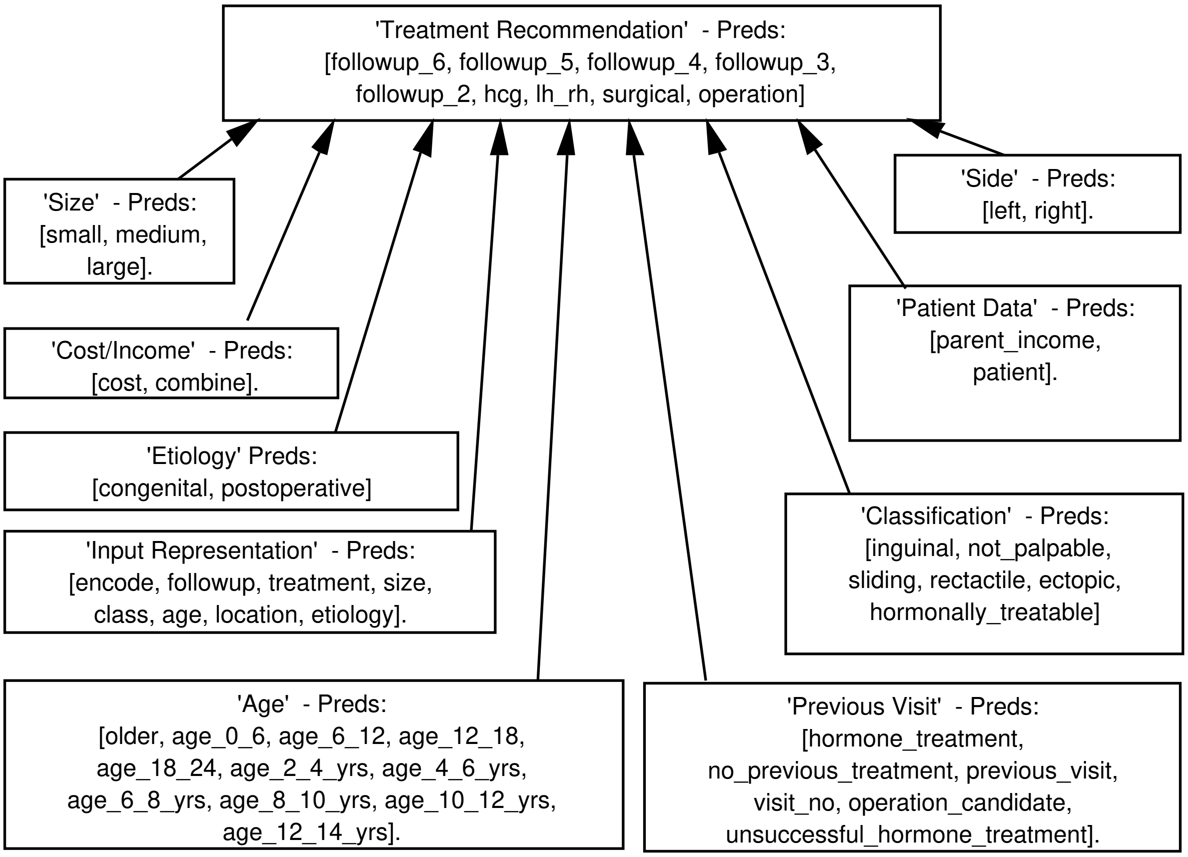

Figure 1: Example of a topology

will be examined by the system to those that are interesting in the specific task context. Furthermore, the direction of the links can be used to reflect the desired direction of inference in the domain. Figure 1 shows the topology used in Maldecensus Testis domain that is further explained in sections 4 and 5. By providing a finely grained topology, a very effective reduction of the hypothesis space is possible. RDT can be used without a topology as well; in this case, $\mathcal{H}$ is the relevant hypothesis space.

Our topology approach is similar in spirit to approaches that allow the specification of irrelevance relationships, such as the one discussed by Subramanian and Genesereth [SG87]. Putting a predicate p in a different topology branch than predicate q effectively tells the system that p is irrelevant for constructing rules about q, or rather, that we are not interested in learning such rules.

4 The RDT algorithm

In this section we will present an efficient algorithm to learn all valid nonredundant rules which are instances of rule models (see last section). It is not required to learn rules that cover all examples[10], nor that each example is covered only once.

[10] CLT, the concept learning tool of MOBAL, has this global view and checks the quality of the rule set [Wro88].

The hypothesis space, defined by the sets of available rule models and predicates and the topology, is searched top-down from general to specific. The search is along the generality hierarchy with pruning of specializations of accepted and failed hypotheses. Acceptance and failure are decided with a user-settable acceptance criterion.

Further effectiveness comes from the use of a many sorted logic. The sort taxonomy tool STT [Kie88] automatically builds up a lattice of sorts. Thus, the user does not need to define the sorts — instead, the sort lattice is created by the system based on the input facts. In the worst case, building the lattice with intersections takes time and space exponential in the number of sorts. Once the sort taxonomy is built, however, it reduces the size of the hypothesis space very effectively, since only well-sorted hypotheses are generated. This enhances the efficiency of learning.

In this section we first explore the generality structure between rule models to improve the search in the hypothesis space. Second, we define an order of the premises within a single rule model to make the pruning mechanism applicable to partially instantiated rule models. Third, we show how these structures are used by the RDT algorithm to prune the search. The specializations of both unacceptable hypotheses and already accepted hypotheses are pruned: the former because they are surely wrong, and the latter because they are redundant. Acceptability of hypotheses is determined using an acceptance criterion which can be set by the user. Finally, we present a pseudocode summary of the algorithm.

4.1 Exploiting the generality structure of the rule models

Since RDT is a model-driven algorithm, the hypothesis space is defined by the set of rule models. This means we do not have a specialization operator as e.g. MIS [Sha83] or FOIL [Qui90] that incrementally builds the hypothesis space to be searched. Instead MOBAL's model knowledge can be partially ordered by an extended form of θ-subsumption[Plo70] defined below. The proposed algorithm searches this partial ordering from the most general to the more specific hypotheses. A breadth first search strategy is used to avoid testing hypotheses that are later subsumed by more general ones (this could happen if one were to use depth first search). The hypotheses that have already been confirmed or pruned as too special (the leaves of the search, see below) are remembered and checked to avoid exploring their specializations.

Suppose for example the rule model developed to learn in the ICS/FORTH domain Maldecensus Testis (see next section and [Wro90]). This rule model was constructed for learning in domains that consist largely of feature vector data. This set consisted of 10 rule models (rule schemes) of increasing complexity:

Q(X) $\leftarrow$ P1(X)
Q(X) $\leftarrow$ P1(X), P2(X)
Q(X) $\leftarrow$ P1(X), P2(X),P3(X)
Q(X) $\leftarrow$ P1(X), P2(X), P3(X), P4(X)

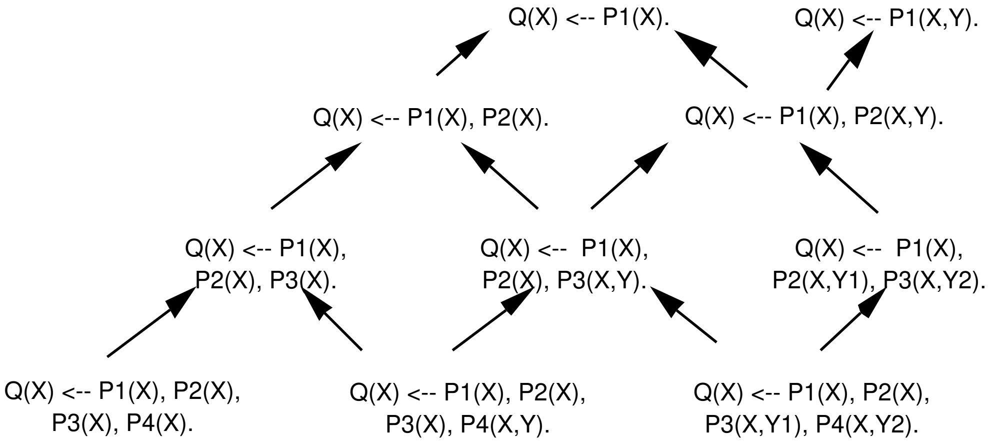

Figure 2: Generalization Relationships between rule models

Q(X) $\leftarrow$ P1(X,Y)
Q(X) $\leftarrow$ P1(X), P2(X,Y)
Q(X) $\leftarrow$ P1(X), P2(X), P3(X,Y)
Q(X) $\leftarrow$ P1(X), P2(X,Y1), P3(X,Y2)
Q(X) $\leftarrow$ P1(X), P2(X), P3(X), P4(X,Y)
Q(X) $\leftarrow$ P1(X), P2(X), P3(X,Y1), P4(X,Y2)

For an efficient search through the hypothesis space and to prevent redundant hypotheses, we have to arrange this unrelated set of rule models with respect to a generality relationship that allows us to prune the search (see Figure 2). This generality relation is trivial to compute if the available background knowledge is not taken into account. If we take for a moment the P_i as predicates and not as predicate variables then we could use θ-subsumption from [Plo70], i.e., a hypothesis H θ-subsumes a hypothesis H' (in clausal form), $H \geq_\theta H'$, iff there is a substitution σ such that $H\sigma \subseteq H'$.

But because the P_i are predicate variables, the situation is a little bit more complicated. First of all we should clearly say what we mean by the generality relationship in the case of rule schemata with predicate variables. Clearly, we do not mean what θ-subsumption originally means, i.e. if H is true (in the intended interpretation) and $H \geq_\theta H'$ then H' must be true (in the intended interpretation) as well. This is because rule models can be neither true nor false, only instantiations of rule schema where all predicate variables are replaced with predicates can be true or false. These considerations lead directly to the definition of our generality relationship $\geq_{RS}$ between rule models.

> If there exists an instantiation $R\Sigma$ of a rule model R and R is more general than R' $R \geq_{RS} R'$, then there exists an instantiation $R'\Sigma$ of R' and $R\Sigma \geq_\theta R'\Sigma$ [11].

[11] This presupposes that R' can in principle be instantiated, i.e. for each predicate variable there exists at least one predicate of the same arity.

This definition explains how $\geq_{RS}$ can be used to specialize hypotheses. But we want to compute $\geq_{RS}$, without looking at instantiations. Therefore, in addition to the substitution applied to term variables as used in the definition of $\geq_\theta$ we use a substitution Σ applied to predicate variables to rename or instantiate the predicate variables. Note that a substitution σ applied to term variables always leads to a specialization (i.e. for each σ: R $\geq_{RS}$ Rσ), whereas a substitution applied to predicate variables could lead to a generalization (i.e. R$\Sigma \geq_{RS}$ R) if this substitution unifies some predicate variables. For instance,

P(X,Y)σ = P(X,X), with $\sigma = Y/X$ and P(X,Y) $\geq_{RS}$ P(X,X).

On the other hand,

$P1(X,Y), P2(X,Y)\Sigma = P1(X,Y)$, with $\Sigma = P2/P1$ and
$P1(X,Y) \geq_{RS} P1(X,Y), P2(X,Y)$.

With these considerations in mind it is easy to see that the following holds and could be used to effectively compute $\geq_{RS}$:

R $\geq_{RS}$ R', iff there is a substitution σ applied to term variables, and a substitution Σ applied to predicate variables that does not unify different predicate variables, such that $R\sigma\Sigma \subseteq R'$.

If R $\geq_{RS}$ R', then at least one such predicate variable substitution Σ exists. For each substitution Σ there exists exactly one substitution σ such that $R\sigma \subseteq R'\Sigma$. But, possibly more than one substitution Σ exists. If more than one substitution Σ exists, we can distinguish two cases:

- Substitutions which lead to identical specializations except for variable names.
- Substitutions which lead to different specializations.

Suppose for example we have

R = {¬P(X), ¬R(X,Y), Q(Y)}, and
R'= {¬P11(X1), ¬P12(X1), ¬R1(X1,Y), ¬P21(X2), ¬R2(X2,Y), Q(Y)},

then

Σ = {P/P11, R/R1} and
Σ = {P/P12,R/R1}

lead to identical specializations with σ = {X/X1}, whereas

Σ = {P/P21, R/R2}

leads to a different specialization with $\sigma = \{X/X2\}$.

This differentiation is not important as long as R does not contain any predicate symbols. But if R is partially or fully instantiated, i.e., R contains predicate symbols, the resulting specializations are different.

4.2 The order of premises within a rule model

The $\geq_{RS}$ order on rule models builds a structure of the hypothesis space which is used for pruning the search. But this is only a kind of a macro structure. There are still many possible hypotheses for a single rule model, namely, all instantiations of the predicate variables of the rule model with predicate symbols. This section is concerned with a fine structuring of the space of hypotheses for a single rule model. This allows to prune the search of hypotheses also within a single rule model. Suppose we have a rule model with two premises, and we do not find a predicate for the first premise which fulfills our acceptance criterion. We then need not look for predicates for the other premises. This requires, however, an order on the premises which respects the variable bindings of the rule model. An example may illustrate this. Suppose the rule model is

$$C(X_1) \leftarrow P(X_2), R(X_1, X_2).$$

It does not make sense to look for a predicate for P before we have found one for R. Every predicate could be used for P without any effect on the acceptance criterion as long as we have not related the variable X_2 to the variable of the conclusion via the relation R. Similar problems occur in other FOL learning algorithms. In FOIL or MIS, this problem is solved by adding only those literals to the premises, which share one variable with the other premises or the conclusion.

To solve this problem for RDT, we first define the *connection* of a variable to the conclusion. Based on the connection, we are able to define a measure for the distance of a variable from the conclusion. This measure is then used to determine the premise order.

The connection of a variable X to the conclusion via the relation chain rc(X) is defined as follows:

- A variable X occurring in the conclusion of a rule model is connected via the empty relation chain (rc(X)=$\emptyset$).
- A variable X_i ($1 \leq i \leq n$) occurring in a premise $R(X_1,X_2,\ldots,X_n)$ is connected via the relation chain $rc(X_i) = R \circ rc(X_j)$[12], iff a variable X_j ($1 \leq j \leq n$), $i \neq j$ of $R(X_1,X_2,\ldots,X_n)$ is connected via the relation chain $rc(X_j)$.

A variable can have more than one relation chain. But a rule model which contains an unconnected variable is not allowed.

The distance of a variable X is then defined as the length of the minimal relation chain connecting it to the conclusion.

$$\delta(X) = \min(\{\mathrm{length}(rc(X)) \mid rc(X) \text{ is a connection relation chain for } X\}).$$

The order of premises is then defined using the distances of the variables occurring in it.

[12] $\circ$ denotes list concatenation, and $R \circ \emptyset = R$.

P $\leq_P$ P', iff min($\{\delta$(X) | X occuring in P$\}$) $\leq$ min($\{\delta$(X) | X occurring in P'$\}$.

If we instantiate the premises of a rule schema with respect to this order, i.e. if P $\leq_P$ P' then P is instantiated before P', it makes sense to test all partial hypotheses (the partial instantiations of a rule model). This testing can be done by the same procedure as the test of the fully instantiated rule model, by simply dropping all uninstantiated premises. Incremental instantiation (with dropping of uninstantiated premises) produces a sequence of rules $r_1, r_2, \ldots, r_n$ that are in θ-subsumption order, i.e. $r_1 \geq_\theta r_2 \geq_\theta \ldots \geq_\theta r_n$. So the same kind of pruning that applies to rule models along the $\geq_{RS}$ order can also be applied incrementally to partially instantiated rule models.

4.3 Pruning the search through incremental testing

The search in the hypothesis space is pruned in several ways. First, the predicate topology and sort lattice are used to avoid considering unrelated predicates for specializations. In addition, the RDT algorithm will not search for specializations of hypotheses that already have too few instances (because the specializations could never be confirmed), nor for specializations of those hypotheses that are confirmed (since their specializations are redundant). The threshold for too few instances is computed from the user specified acceptance criterion that is defined as follows. Let H be a possible hypothesis:

$$\mathrm{H} = \mathrm{q}(\mathrm{X}_1, \cdots, \mathrm{X}_\mathrm{n}) \leftarrow \mathrm{p}(\mathrm{X}_1, \cdots, \mathrm{X}_\mathrm{m})$$

where $m \geq n$, and $p(X_1, \cdots, X_m)$ represents the conjunction of the premises $p_1, \cdots, p_j$ over the variables $X_1, \cdots, X_m$. Then the primitives for the user-settable acceptance criterion are defined as the cardinalities of:

- pos(H) := $\{(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{n}) | \mathrm{p}(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{m}) \& \mathrm{q}(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{n})\}$, the positive instances of H
- neg(H) := $\{(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{n}) | \mathrm{p}(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{m}) \& \mathrm{not}(\mathrm{q}(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{n}))\}$, the negative instances of H
- pred(H) := $\{(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{n}) | \mathrm{p}(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{m}) \& \mathrm{unknown}(\mathrm{q}(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{n}))\}$, the unknown, i.e. neither provable true nor provable false, instances of Q which will be predicted by H.
- total(H) := $\{(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{n}) | \mathrm{p}(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{m})\}$ = pos(H) $\cup$ neg(H) $\cup$ pred(H), the total instances of H.
- unc(H) := $\{(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{n}) | \mathrm{q}(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{m})\} \setminus \{(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{n}) | \mathrm{p}(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{m})\}$, the instances of Q which are uncovered by H .
- concl(H) := $\{(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{n}) | \mathrm{q}(\mathrm{c}_1, \cdots, \mathrm{c}_\mathrm{m})\}$, all instances of the conclusion Q.

The acceptance criterion is a logical expression of conjunctions and disjunctions of arithmetical comparisons (i.e. =, <, $\leq$, >, $\geq$) involving arithmetical expressions (i.e. +, -, *, /) built from numbers and the above primitives. An example is:

pos > 5, neg < 2, pos/total > 0.7, pred/pos > 0.3, unc/concl < 0.5

For the acceptance criterion primitives the following relations to θ-subsumption $\geq_\theta$ hold: Let H and H' be two hypotheses and H $\geq_\theta$ H' then

1. pos(H) $\supseteq$ pos(H').
2. neg(H) $\supseteq$ neg(H').
3. pred(H) $\supseteq$ pred(H').
4. total(H) $\supseteq$ total(H').
5. unc(H) $\subseteq$ unc(H').
6. concl(H) = concl(H').

Using these relations between the primitives and $\geq_\theta$ together with the semantics of the expression building operators, a pruning criterion is derived from the acceptance criterion. This derived pruning criterion only prunes hypotheses, which cannot fulfill the acceptance criterion, i.e. if a hypothesis h fulfills the pruning criterion, than no specialization of h will fulfill the acceptance criterion. For our example the derived pruning criterion is:

pos $\leq$ 5, unc/concl $\geq$ 0.5

In normal expressions the qualities 1, 4 and 5 can be used to prune the search[13]. If a hypothesis H does not fulfill the acceptance criterion with respect to pos, total or uncovered, then all specializations of H will not either. On the other hand, if a hypothesis H fulfills the acceptance criterion, then we need not look at the specializations of H. This step also prevents the addition of redundant conditions.

In summary, the use of a pruning criterion as described here still guarantees that the algorithm will be complete with respect to the hypothesis space $\mathcal{H}^{\mathcal{T}}$, i.e. it will find all hypotheses in $\mathcal{H}^{\mathcal{T}}$ that are acceptable according to the chosen acceptance criterion, and that are not subsumed by other accepted hypotheses.

4.4 Summary of the algorithm

Figure 3 is a pseudocode summary of the algorithm that was described above. The algorithm is called using the command `rdt(Q)`, where Q is a literal with the target predicate for which the RDT algorithm is to learn all possible rules. If the

[13]In principle the user could build any expression, but it is very unlikely that he uses something like neg > 2 (i.e. to accept the hypothesis it must have more than two negative examples)

```
rdt(Q)
   set RS to {}
   set LEAVES to {}
   for all known rule models R
      if the conclusion C of R is unifiable with Q
      then push RΣ, where Σ = {C/Q} onto RS
   while RS ≠{}
         pop a most general (with respect to ≥_RS) rule model R from RS
         instantiate-and-test(R,TOO-GENERAL)
         for all X element RS which are specializations of R
            pop X from RS
            for all Y element TOO-GENERAL
               for all really different Σ: Yσ ≥_RS XΣ)
                  push XΣ onto RS
            endfor
         endfor
   endwhile

instantiate-and-test(HYPO,TOO-GENERAL)
   while HYPO ≠ {}
         pop a hypothesis H from HYPO
         select a smallest (with respect to ≤_P) uninstantiated premise PREM of H
         for all predicates P which are
                  (arity-compatible with PREM) and
                  (topology-compatible with the conclusion of H) and
                  (sort-compatible with H)
            set H to HΣ with Σ = {PREM/P}
            if the instantiated part of H is not a specialization (≥_θ)
              of an element of LEAVES
            then test the instantiated part of H against the acceptance criterion
                  if H is not to special (i.e. is acceptable with
                                          respect to pos, total and unc)
                  then if all premises of H are instantiated
                        then if H is acceptable with respect to all criteria
                              then push H onto LEAVES
                                    assert H in the KB
                              else push H onto TOO-GENERAL
                        else push H onto HYPO
                  else push H onto LEAVES
            fi
         endfor
   endwhile
```

Figure 3: The RDT algorithm

literal is negative, the algorithm learns the negation of the target predicate based on the available members of $\mathcal{R}$ with negative conclusions.

The hypothesis queue `RS` and the list of leaves `LEAVES` together represent the state of the search. This makes it easy to suspend and resume the search and to learn rules for several concepts in parallel.

Note that the generalization hierarchy and the premise order are independent of the learning task. They depend only on the available rule models. This means that both could be precomputed, and the time for building them need not be spent again for each learning task. So a format that supports quick access to the most general rule models, to all specializations of a rule model together with the possible matches (i.e. Σ), and to the ordered premises could be used without extra cost for learning.

Further optimizations are also possible in the hypothesis test step by remembering the instances of the parent hypothesis (as in FOIL). In this case, we only need to check the newly added premises against this instance set instead of checking the entire hypothesis against the entire knowledge base.

The test against the acceptance criterion is based on the user-settable parameter. The building blocks for this criterion are discussed in the section about incremental testing above.

5 RDT and MOBAL

RDT as described in section 4 is an algorithm that learns rules for a specified target concept or relation. In this section, we want to briefly describe the use of RDT within the knowledge acquisition system MOBAL [Wro89], a successor of BLIP [EKK+89]. MOBAL is a system that integrates manual knowledge acquisition and machine learning according to the so-called *sloppy modeling* paradigm [Mor89, Mor90]. MOBAL provides a comfortable and graphical interface for the entry and manipulation of facts, rules, rule models, and topology (among others). RDT is used in three ways within the MOBAL system:

- The user may call RDT and explicitly provide the target predicate.
- The concept learning tool CLT calls RDT to learn rules[14] about a proposed concept, and then evaluates their quality as a concept description [Wro88].
- RDT is used as a background learning process that continuously tries to induce rules and include them in the knowledge base.

In the latter use, a set of heuristics is used to control which target predicate RDT is working on at a given moment.

[14] The version of RDT presented here will only find rules that use the target concept as conclusion, i.e., sufficient conditions. The previous version of RDT was also able to find all rules that used the target concept in the premises.

Within MOBAL, RDT has been applied to a number of toy and real-world problems[15]. In [Wro90], we describe the application of MOBAL to the medical domain *Maldecensus Testis* that we are developing together with ICS/FORTH of Greece. In this application, rules about the proper treatment of infants and children with a certain testicle problem are to be learned. In a first trial based on 50 (highly biased) cases, RDT induced a number of rules about the required followup period for a patient that were plausible from a medical point of view, but not perfectly satisfying yet. We have also begun work on the diagnosis of time-dependent fault states in satellites, a domain on which CIGOL has already been tried [Fen90].

6 Future Research

At present, the version RDT described here is being implemented and integrated into the MOBAL system. From our experiences with the predecessor version of RDT contained in MOBAL-0, we have already identified a number of issues that could still be addressed in future versions of RDT:

- Learning rules with constants and computed predicates
- Automatic construction of a topology
- The integration of heuristic search control criteria
- Means of automatically extending the set of rule models $\mathcal{R}$.

Constants and computed predicates In many domains with attributive or relational data, it is very important to have the facility to learn rules with constants and computed comparison predicates, such as:

no_parking(X) $\leftarrow$ place_description(X,Y) & eq(Y,bus_lane)
fever(X) $\leftarrow$ human(X) & body_temperature(X,T) & gt(T,37.5)

In keeping with the model-based spirit of RDT, the user must be allowed to supply rule schemata for rules such as the ones above, i.e., including computed predicates and distinguished arguments where the system is to search for constants. Since RDT already allows rule schemata whose predicate variables are partially pre-instantiated with domain predicates, computed predicates can be included simply by using their (reserved) names in the rule schema. The argument positions in which the system is to look for constants can be identified eg. by adopting the convention that all variables that occur only in computed predicates are candidates for replacement by constants, or by allowing the user to explicitly mark such argument places. If the former convention is used, the rule schemata for the example rules above would have to be:

[15] Those applications have used MOBAL-0, the first version of MOBAL, which contains an older version of RDT that uses rule models and a topology, but does not use the search control algorithm described in section 4.

$Q(X) \leftarrow R(X,Y), eq(Y,T).$
$Q(X) \leftarrow C(X), R(X,Y), gt(Y,T).$

In this example R,C and Q are predicate variables and T is a variable for a constant to learn.

To use such rule schemata for learning, RDT needs to recognize the names of computed predicates, and call their evaluation procedures instead of looking in the knowledge base. For constants, the appropriate procedure depends on the argument in which the constant is to be substituted. For the computed predicate "eq" (and for constants in non-computed predicates), all possible remaining instantiations of the variable must be tried; for comparison predicates on ordered values like "ge" (greater than or equal) etc., special techniques such as the ones described in [BFOS84] for attribute-value systems will be useful.

To make the process of looking for constants maximally efficient, it needs to be properly integrated into RDT's incremental process of instantiating and testing partial rule schemata. Because there are in most cases many more constants than predicates, RDT should look first for the predicates, and then look for the remaining constants, i.e. constants should be incorporated in the premise order after all premises.

Topology construction Currently under work is a tool for the automatic construction of the topology [Kli91]. The automatically constructed topology is an abstraction of the rules in the knowledge base, i.e. it is built from user given and previously learned rules. Such an automatically constructed topology can be used to avoid having to construct a topology manually, or it can be used as the basis for further manual improvement.

Heuristic search control One of the simplest, but nevertheless powerful, improvements is the use of an information theoretic criterion like the one in FOIL combined with best-first selection of specializations of rule schemata. This seems to be a very promising direction, since our method of specialization based on rule schemata allows us to overcome the myopy problem of FOIL, since we can introduce more than one conjunct at a time in a controlled fashion.

Automatic extension of $\mathcal{R}$ MOBAL contains a tool (MAT) for the abstraction of a rule schema from user given rules. This tool could be used to automatically acquire new rule schemata in cases where the current ones are not sufficient for learning by using a non-model driven learning method to search the "gaps" left by the existing model. This could be done by any learning method capable of dealing with RDT representation (function free horn clauses), eg. ARCH, INDUCE, FOIL, MIS. Any rules found by this more complete search would then be abstracted by MAT and added to the rule model. This approach has to be combined with a usefulness criterion and the deletion of useless rule schemata.

Most promising in this direction seems to be the direct integration of RDT and a not model-based learner which also searches top-down (e.g. FOIL). In such an approach, the efficiency of RDT is used for scanning the search space. This search is done much faster than a not model-based algorithm could do. But if RDT fails to learn, the most promising hypothesis could be further refined by the other algorithm. Such an integrated approach combines two advantages: the efficiency of model-based learning and the flexibility of other algorithms. In addition, it overcomes the restriction of model-based learning, namely that well-suited models need to be given. If needed, new rule models can be found by the "backup" algorithm. In addition, this approach overcomes the restriction of other algorithms, namely the short-sightedness of heuristic search. These algorithms now only need to learn within a subarea of an already scanned hypothesis space. Therefore, they do not rely so much on their heuristics but can search this subspace more completely.

Acknowledgements

This work was partially supported by the European Community ESPRIT program under contract number P2154 "Machine Learning Toolbox". MOBAL is developed as a team effort of Christian Haider, Jörg-Uwe Kietz, Volker Klingspor, Katharina Morik (head), and Stefan Wrobel. The authors especially wish to thank Katharina Morik for important contributions during the design phase of the new RDT, and for reading and commenting earlier drafts of this paper.

System Availability

The MOBAL system (of which RDT is one part) is implemented in Quintus Prolog under SUN Unix; its user interface is implemented in HyperNeWS. For non-commercial use, the system is available from GMD for free.

References

[BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. *Classifikation and regression trees.* Belmont, Wadsworth, 1984.

[BS85] Ronald J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9, 1985.

[Bun88] Wray Buntine. Generalized subsumption and its applications to induction and redundancy. *Artificial Intelligence*, 36:149 – 176, 1988.

[CGT90] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer Verlag, Berlin, New York, 1990.

[DB89] L. DeRaedt and M. Bruynooghe. Towards friendly concept-learners. In *Proc. of the 11th Int. Joint Conf. on Artif. Intelligence*, pages 849 – 854, Los Altos, CA, 1989. Morgan Kaufman.

[dRB89] Luc de Raedt and Maurice Bruynooghe. Constructive induction by analogy: a method to learn how to learn? In *Proc. Fourth European Working Session on Learning (EWSL)*, pages 189 – 199, London/Los Altos, CA, 1989. Pitman/Morgan Kaufman.

[EKK+89] Werner Emde, Ingo Keller, Jörg-Uwe Kietz, Katharina Morik, Sabine Thieme, and Stefan Wrobel. Abschlussbericht des BMFT-Projektes 3 b LERNER. KIT-Report 71, Technische Universität Berlin, D-1000 Berlin, West Germany, April 1989. (English book version in preparation).

[Emd87] Werner Emde. Non-cumulative learning in METAXA.3. In *IJCAI-87*, Los Altos, CA, August 1987. Morgan Kaufman.

[Emd89a] Werner Emde. An inference engine for representing multiple theories. In Katharina Morik, editor, *Knowledge Representation and Organization in Machine Learning*, pages 148–176. Springer Verlag, Berlin, New York, 1989. auch: KIT-Report Nr. 64, TU Berlin, 1988.

[Emd89b] Werner Emde. Lernen im geschlossenen Kreislauf. In W. Brauer, editor, *3. Internationaler GI-KongreßWissensbasierte Systeme (SYSTEMS)*, Informatik Fachberichte, Berlin, New York, 1989. Springer Verlag.

[Fen90] Cao Feng. Application of CIGOL for fault diagnosis for the power subsystem of Skynet satellite. MLT Technical Note, The Turing Institute, Glasgow, Scotland, May 1990.

[Kie88] Jörg-Uwe Kietz. Incremental and reversible acquisition of taxonomies. *Proceedings of EKAW-88*, pages 24.1–24.11, 1988. Also as KIT-Report 66, Technical University Berlin.

[Kli91] Volker Klingspor. Abstraktion von Inferenzstrukturen in MOBAL. Master's thesis, Univ. Bonn, 1991. In German. To appear.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, Berlin, New York, 2nd edition, 1987.

[MB88] Stephen Muggleton and Wray Buntine. Machine invention of first-order predicates by inverting resolution. In *Proc. Fifth Intern. Conf. on Machine Learning*, Los Altos, CA, 1988. Morgan Kaufman.

[MF90] Stephen Muggleton and Cao Feng. Efficient induction of logic programs. In *Proc. First Conf. on Algorithmic Learning Theory*, Tokyo, 1990. Ohmsha Publishers.

[Mic83] Ryszard S. Michalski. A theory and methodology of inductive learning. In *Machine Learning — An Artificial Intelligence Approach*, pages 83 – 134. Morgan Kaufman, Los Altos, CA, 1983.

[Mit82] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203 – 226, 1982.

[Mor89] Katharina Morik. Sloppy modeling. In Katharina Morik, editor, *Knowledge Representation and Organization in Machine Learning*, pages 107–134. Springer Verlag, Berlin, New York, Jan. 1989.

[Mor90] Katharina Morik. Integrating manual and automatic knowledge aquisition - BLIP. In Karen L. McGraw and Christopher R. Westphal, editors, *Readings in Knowledge Acquisition – Current Practices and Trends*, chapter 14, pages 213 – 232. Ellis Horwood, New York, 1990.

[Mug90] Stephen Muggleton. Inductive logic programming. In *Proc. First Conf. on Algorithmic Learning Theory*, Tokyo, 1990. Ohmsha Publishers.

[Plo70] Gordon D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence 5*, chapter 8, pages 153 – 163. Edinburgh Univ. Press, Edinburgh, 1970.

[Qui83] J. Ross Quinlan. Learning efficient classification procedures and their application to chess end games. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning — An Artificial Intelligence Approach*, pages 463 – 482. Tioga, Palo Alto, CA, 1983.

[Qui90] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3):239 – 266, 1990.

[RP89] Céline Rouveirol and Jean François Puget. Beyond inversion of resolution. In *Proc. Sixth Intern. Workshop on Machine Learning*, pages 122 – 130, Los Altos, CA, 1989. Morgan Kaufman.

[SB86] Claude Sammut and Ranan B. 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 II, chapter 7, pages 167 – 191. Morgan Kaufman, Los Altos, CA, 1986.

[SG87] Devika Subramanian and Michael Genesereth. The relevance of irrelevance. In *Proc. 10th International Joint Conference on Artificial Intelligence*, pages 416 – 422, Los Altos, CA, 1987. Morgan Kaufman.

[Sha83] Ehud Y. Shapiro. *Algorithmic Program Debugging.* ACM Distinguished Doctoral Dissertations. The MIT Press, Cambridge, Mass., 1983.

[Win75] P. H. Winston. Learning structural descriptions from examples. In P.H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.

[Wir89] Ruediger Wirth. Completing logic programs by inverse resolution. In Katharina Morik, editor, *Proc. Fourth European Working Session on Learning (EWSL)*, pages 239 – 250, London/San Mateo, CA, 1989. Pitman/Morgan Kaufmann.

[Wro88] Stefan Wrobel. Automatic representation adjustment in an observational discovery system. In D. Sleeman, editor, *Proc. of the 3rd Europ. Working Session on Learning*, pages 253 – 262, London, 1988. Pitman.

[Wro89] Stefan Wrobel. MLT Deliverable 4.0/G: Description of MOBAL. GMD (German Natl. Research Center for Computer Science), P.O.Box 1240, W-5205 St. Augustin 1, Germany, September 1989.

[Wro90] Stefan Wrobel. Application of MOBAL to the medical domains of ICS/FORTH (MLT technical note). GMD (German Natl. Research Center for Computer Science), P.O.Box 1240, W-5205 St. Augustin 1, Germany, September 1990.