Algebraic Foundation and Improved Methods of Induction of Ripple Down Rules*

Tobias Scheffer
Technische Universität Berlin, Artificial Intelligence Research Group (KI),
Sekr. FR 5-8, Franklinstr 28-29, D-10587 Berlin, Germany
email: scheffer@cs.tu-berlin.de

December 5, 1996

Abstract

Ripple down rules (RDR), that is rules with hierarchical exceptions, are used in knowledge acquisition because they provide a well intelligible and modifiable representation for even very large expert systems. In this paper a formal semantics for RDRs is proposed, that covers first order rules as well as attribute-value based rules. An algebraic foundation is proposed, including simplification of RDRs and transformation of RDRs into flat lists of rules and ripple down rule sets, hence these knowledge representation schemes are put into perspective. It is shown, that a RDR has a shorter description length than an equivalent decision list.

Induction of rules with exceptions is characterized as bidirectional movement in the hypothesis space, while known algorithms for learning rules or decision trees either perform a top-down specialization of the most general or a bottom-up generalization of the most special hypothesis. Known algorithms for induction of RDRs are summarized and compared and a bidirectional algorithm is proposed, that handles continuous-valued attributes using an implication based generalization for attribute-value based representations, and shows good empirical results.

1 Introduction

The process of knowledge acquisition is the largest part of effort in building and maintaining expert systems. Two important aspects of this process addressed in this paper are representation and modularization of knowledge, and support of this process by machine learning algorithms.

To improve the comprehensibility of large knowledge bases, a number of representational schemes was developed that provide modularity by supporting *exceptions* [Ver80, Riv87, KMU93b, HSW89, DK95], the most general for attribute-value based representations and most strongly structured scheme being *ripple down rules* [CJ88]. A ripple down rule (RDR) is a list of rules; each rule may be associated to another list of rules: its exceptions. The implied concept of locality (an exception is applicable only if its next-general rule is applicable) makes RDRs comfortable for human needs: rough rules can be expressed first, the exceptions of which can be modeled as a refinement of the hypothesis later. *Ripple down rule sets* (RDR-sets) [KMU93a], also called *multiple classification ripple down rules* (MC-RDR) [KCP95], differ from RDRs in that rules in a list of successors may fire synchronously, while in a RDR, each rule suppresses its successors. Decision lists [Riv87] contain rules that suppress their successors – hence negation and exceptions can be expressed – but there is no distinction between rules, the instances of which are proper subsets, and rules that partially overlap.

For various examples, it could be shown empirically that Ripple-Down-Rules take less rules to describe complex systems than are needed by a "flat" list of rules [CPKY94, GC92, PEC94, CJ88].

*Proc. Pacific Knowledge Acquisition Workshop, Sydney, 1996. (revised version)

This is evident, because if overly general rules can be specialized by negative conditions, in many cases this results in a shorter description length than is obtained by breaking down such rules, and covering the positive instances by a number of more special rules.

In the area of machine learning, a number of algorithms were developed, that are based on hypothesis languages that include exceptions. For first order hypothesis languages Vere [Ver80] proposed a notation of "multilevel counterfactuals", that is quite similar to the RDR notation, as well as a learning algorithm, most later approaches are based on or are similar to; Wrobel [Wro88] introduces the invention of exception predicates. His learning algorithm does not need to delete rules with only a few counterexamples, as would do Shapiro's debugging system [Sha83] or most ILP algorithms. Bain [BM90] more formally introduces the *most general correct specialization*, a specialization operation that excludes a clause from the domain of another rule, Kijsirikul et al. [KNS92] propose a similar approach to predicate invention.

Section 2 of this paper deals with the semantics and algebraic foundations of RDRs, including the simplification and transformation of RDRs into decision lists and ripple down rule sets. Section 3 deals with the automatic induction of RDRs; known approaches are compared and a new algorithm for induction and revision of RDRs is proposed, that handles continuous attributes.

2 Ripple down rule algebra

Ripple-Down-Rules (RDR), proposed informally in [CJ88], are rules with hierarchically structured exceptions. Fig. 1 shows an example RDR. Lists of rules connected by "if not"-edges behave like decision lists: only if a rule is not applicable to an input object, the next rule is applied. The right-hand-side tree connected to a rule by an "except"-edge contains local exceptions. A rule may fire only if the "except"-tree does not classify the input.

2.1 Requisites

RDRs as originally proposed, discriminate input objects into a set of mutually exclusive classes. Transferred to first-order languages this corresponds to discriminating a set of predicates $P = \{p_i\}$, where $p_i(a)$ implies $not(p_j(a))$ for $i \neq j$, i.e. the predicates are mutually excluding each other for each instantiation.

Each node of a RDR contains a rule which is assumed to be a horn clause of an arbitrary first-order language where a consequence relation $\vdash$ is defined. We assume that this consequence relation ensures the mutual exclusiveness of different predicates, i.e. $p_i(a) \vdash not(p_j(a))$ for $i \neq j$. Possible consequence relations include θ-subsumption modulo background theory B or logic implication. The *domain* of a rule r is the set of literals $p_i(x_1, \ldots, x_n)$, such that r either implies $p_i(x_1, \ldots, x_n)$, or r implies $p_j(x_1, \ldots, x_n)$, i.e. $not(p_i(x_1, \ldots, x_n))$, because we assumed that $p_i(\ldots)$ implies $not(p_j(\ldots))$. Basically, the domain is the set of instantiations, the rule fires for and either proves or refutes the conclusion. It will be required for the definition of some equations for reduction of RDRs.

Definition 1 (Domain) *Let r be a rule, B a background theory and $\vdash$ a consequence relation. Then the* domain *of r, $dom(r)$ is the set of literals $p_i(x_1, \ldots, x_n)$, such that $r \wedge B \vdash p_i(x_1, \ldots, x_n)$ or $r \wedge B \vdash p_j(x_1, \ldots, x_n)$*

2.2 Propositional and first-order rules

Although the following considerations deal with first-order rules, the special case of propositional rules is of particular importance, because propositional representations are frequently used in knowledge acquisition and machine learning and many of the following propositions simplify in this case. Hence, we describe how propositional rules fit into our framework and which of their properties are of importance.

Let $\{(x_i, c_i)\}$ be a data base where x_i is a case description in terms of an attribute vector $(x_i = (x_{i1} \ \ldots \ x_{in}))$ and c_i the corresponding class. Each class c is then represented by a n-ary

predicate c/n, each sample i is represented by a literal $c_i(x_{i1} \ \ldots \ x_{in})$. If we further assume that our rule language consists of interval constraints for the variables occurring in the head, i.e. the rules have the form $c(x_1, \ldots, x_n) \leftarrow x_1 \in [a_1, b_1] \wedge \ldots \wedge x_n \in [a_n, b_n]$, then $r \vdash l$ if $r\theta = l$ for some θ and the interval constraints are satisfied for $r\theta$, which can be tested in linear time. This language has another interesting property: As there are no common variables in any two interval constraints, the domain of a conjunction of two constraints is the intersection of the individual domains.

Proposition 1 *Let* $r = c(x_1, \ldots, x_n) \leftarrow x_1 \in [a_1, b_1] \wedge \ldots \wedge x_i \in [a_i, b_i] \wedge x_{i+1} \in [a_{i+1}, b_{i+1}] \wedge \ldots \wedge x_n \in [a_n, b_n]$. *Then* $dom(r) = dom(c(x_1, \ldots, x_n) \leftarrow x_1 \in [a_1, b_1] \wedge \ldots \wedge x_i \in [a_i, b_i]) \cap dom(c(x_1, \ldots, x_n) \leftarrow x_{i+1} \in [a_{i+1}, b_{i+1}] \wedge \ldots \wedge x_n \in [a_n, b_n])$.

Proof 1 $dom(c(x_1, \ldots, x_n) \leftarrow x_i \in [a_i, b_i]) = \{c_k(y_1, \ldots, y_n) | y_i \in [a_i, b_i]\}$. $dom(c(x_1, \ldots, x_n) \leftarrow x_j \in [a_j, b_j]) = \{c_k(y_1, \ldots, y_n) | y_j \in [a_j, b_j]\}$. $dom(c(x_1, \ldots, x_n) \leftarrow x_i \in [a_i, b_i] \wedge x_j \in [a_j, b_j]) = \{c_k(y_1, \ldots, y_n) | y_i \in [a_i, b_i] \wedge y_j \in [a_j, b_j]\} = dom(c(x_1, \ldots, x_n) \leftarrow x_i \in [a_i, b_i]) \cap dom(c(x_1, \ldots, x_n) \leftarrow x_j \in [a_j, b_j])$. *This is true for any conjunction of constraints, hence for any conjunction of conjunctions.*

2.3 Semantics of RDR

We notate RDRs as triples $\langle rule, X, N \rangle$, where X and N are the exception RDR and the succeeding RDR respectively. The empty RDR is notated λ. More clearly, RDRs are depicted graphically as in fig. 1 with boxes marking rules and edges labeled "except" and "if not" pointing at the nested exceptions and the succeeding RDR.

Definition 2 *A ripple down rule* $R = \langle rule, X, N \rangle$ *yields a result* c, *written* $R \wedge B \vdash_{RDR} c$, *by the following strategy:*

$$\lambda \wedge B \not\vdash_{RDR} c \qquad \text{for any } c \tag{1}$$

$$\langle rule, X, N \rangle \wedge B \vdash_{RDR} c \quad \Leftrightarrow \quad \begin{cases} rule \wedge B \vdash c & and & X \wedge B \not\vdash_{RDR} \bar{c} & (a) \\ or \quad X \wedge B \vdash_{RDR} c & and & rule \wedge B \vdash \bar{c} & (b) \\ or \quad N \wedge B \vdash_{RDR} c & and & rule \wedge B \not\vdash c' & (c) \end{cases} \tag{2}$$

That is, the rule fires and yields c for an observation B (written $rule \wedge B \vdash c$) if (a) the exception RDR does not yield any other result. Else (b) the rule fires but the exception tree's result overshadows the result ($X \wedge B \vdash_{RDR} c$). If (c) the rule does not yield any result for the given observation ($rule \wedge B \not\vdash c'$ for any c'), the result from the next RDR is returned ($N \wedge B \vdash_{RDR} c$).

There may be more than two possible conclusions (c and $\neg c$) occurring in a RDR. In this case, the RDR *classifies* the input, e.g. if there are three classes c_1 to c_3, then a conclusion of c_1 implies $\bar{c}_2$ and $\bar{c}_3$. Thus, here can be at most one conclusion from a RDR and an observation for propositional languages and one conclusion for every instantiation of the head in the non-propositional case.

Fig. 1 shows an example RDR for an informal condition language: The first rule says "birds fly"; it has an "if not"-successor (airplanes fly) and a tree of exceptions: "young birds don't fly"and "penguins don't fly"which in turn has another exception, "penguins in planes fly".

A problem about RDRs is that the rules are implicitly augmented with the negations of all preceding rules, because the succeeding rules may only be applied if the current rule does not apply; this might spoil the good comprehensibility. In [KMU93a] *ripple down rule sets* (RDR-set) are proposed; here all rules in a list of successors may fire unless suppressed by their exceptions. Note that in contrast to a RDR, a RDR-set may yield more than one result even for propositional rules and observations.

According to def. 2 the semantics of a RDR-set can be defined by replacing case (c) of equation 2 by $N \wedge x \vdash c$ and omitting the second part of this condition. Thus, all rules in a successor relation may fire since a rule does not suppress the successor tree. RDRs can be transformed into RDR-sets by explicitly inserting the preceding rules into each rule's exceptions. A RDR-set that computes

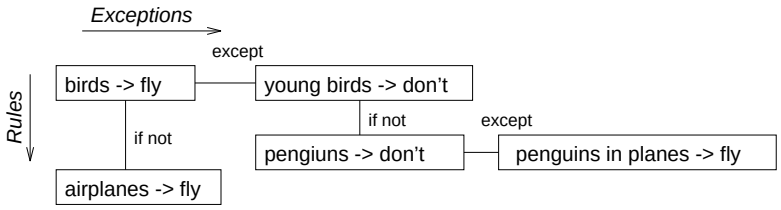


Figure 1: An example Ripple-Down-Rule

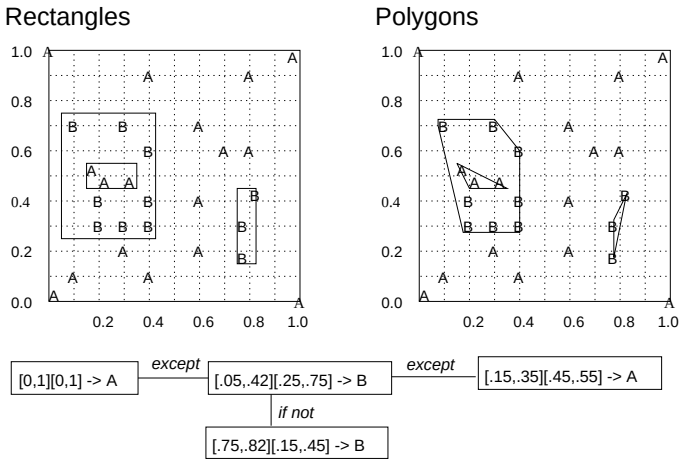


Figure 2: Ripple-Down-Rules in attribute space

at most one conclusion for each observation is a RDR; if one rule only fires, the suppression effect cannot change the semantics.

A concept is set up by a number of rules, each of them is "subtracted" an exception concept. While for RDRs this concept is stated explicitly by a nested RDR, another possibility is to *name* this exception concept and set up a number of new rules describing it. While RDRs allow an individual conclusion for each exception rule, by inventing an exception predicate a subset is subtracted only. This is a difference if more than two conclusions exist. On the other hand, the invention of named predicates allows recursive descriptions of exception concepts: this extends the expressiveness in the case of non-propositional rule languages.

2.4 Geometrical interpretation

Attribute-value based ripple down rules can be interpreted geometrically. A rule characterizes an area of the attribute space. Depending on the condition language, this may be a hyper-rectangle if the condition is a list of intervals. Each rule r_X within the exception tree of some rule r is more special than r, the corresponding hyper-rectangle is thus nested into the more general rectangle of r. Each rule r_N in the list of successors of r is either disjoint or else, due to definition 2, r overshadows r_N. That is, a ripple down rule is a structure of overlapping and nested hyper-rectangles, if conditions are hyper-intervals.

Fig. 2 shows two RDRs for attribute spaces with different condition languages: axis-parallel hyper-rectangles and convex polygons. The textual RDR corresponds to the left hand side picture.

2.5 Identifying nodes

In order to identify rules of a RDR we introduce the notation of paths in the tree structure. A path is a list of x's and n's, locating a node relatively to the root. E.g. a path of $< x, n >$ identifies the successor of the first exception of a RDR. We write $< path > (R) = rule$, if $rule$ is the rule at the node located at the node identified by $path$ within the RDR R.

2.6 Congruence of ripple down rules

In this section a congruence relation for behaviorently equal RDRs is introduced. This relation founds the ripple down rule algebra, that is elaborated in the following.

Definition 3 *Ripple down rules R_1 and R_2 are congruent, $R_1 \equiv R_2$, iff $\forall B : R_1 \wedge B \vdash_{RDR} c \leftrightarrow R_2 \wedge B \vdash_{RDR} c$. If this only holds for a certain set of observations $c \in O$, we write $R_1 \equiv_O R_2$.*

That is, two RDRs are congruent ($\equiv$), if they provide identical conclusions for any background theory B, or locally congruent ($\equiv_O$), if if they provide identical conclusions for some particular domain O.

Proposition 2 *$\equiv$ is a congruence relation, i.e. $\equiv$ is reflexive, transitive, symmetrical and compatible with the constructor $\langle \cdot, \cdot, \cdot \rangle$.*

Proof 2 *Definition 3 trivially implies that $\equiv$ is reflexive, transitive and symmetrical. Let $R_1 \equiv R_2$ and let B be any background theory. Compatibility with the constructor is proven, if this implies $\langle r, R_1, R \rangle \equiv \langle r, R_2, R \rangle$ and $\langle r, R, R_1 \rangle \equiv \langle r, R, R_2 \rangle$, which directly follows from definition 2.*

In the following sections, a number of partial equations are proven, that provide the foundations for reduction of RDRs (α and β), additional reduction of propositional RDRs (γ), transformation into decision lists (β') and transformation into RDR-sets (δ). Two further concepts, called *context* and *scope* of a RDR-node, are required to express these equations. The context is the set of observations that reach that node when being classified, the scope is the set of nodes the rule fires for and no exception or preceding rule fired for. While the domain is defined for rules, context and scope are defined for nodes of a RDR, which are identified by paths as introduced in section 2.5.

Definition 4 (Context) *The context of a node p of a RDR R relative to a set of observations O, con_O, is defined as follows:*

1. $con_O(R, <>) = O$
2. $con_O(R, < path, x >) = con_O(R, < path >) \cap dom(rule)$, *where* $< path > (R) = rule$.
3. $con_O(R, < path, n >) = con_O(R, < path >) \setminus dom(rule)$, *where* $< path > (R) = rule$.

The scope is computed by subtracting the domains of all exceptions from the intersection of the context and domain of a node.

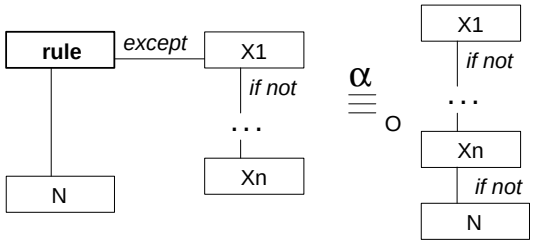
Definition 5 (Scope)

$$\begin{aligned} scope_O(R, < path >) \quad = \quad & con_O(R, < path >) \cap dom(< path > (R)) \\ & \setminus dom(< path, x > (R)) \\ & \setminus dom(< path, x, n > (R)) \setminus \ldots \\ & \setminus dom(< path, x, n \ldots, n > (R)) \end{aligned}$$

2.7 Reduction of ripple down rules

The following partial equation, named α-rule, provides the foundation of an algorithm that eliminates redundant nodes.

Proposition 3 (α-rule)



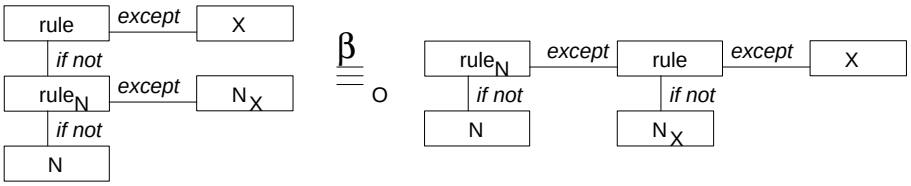
iff $scope_O(R, < pos >) = \emptyset$ *and* $dom(rule) \supseteq dom(X_1)$ *and ... and* $dom(rule) \supseteq dom(X_n)$.

That is, a rule with an empty scope can be deleted, this does not change the behavior of the RDR with respect to those samples, the context was computed relatively to.

Proof 3 *Let $c \in O$ be any literal. (a) $c \notin dom(rule)$. Then due to definition 2 $\langle rule, X, N\rangle \wedge B \vdash_{RDR} c$ iff $N \wedge B \vdash_{RDR} c$. Proposition 3 ensures, that $dom(rule) \supseteq dom(X_i)$ for all X_i, i.e. if $c \notin dom(rule)$ then $c \notin X_i$. Hence, at the right hand side RDR none of the X_i rules fires and $\langle X_1, \ldots X_n, N\rangle \wedge B \vdash_{RDR} c$ iff $N \wedge B \vdash_{RDR} c$, both RDRs return the same result obtained by N. (b) $c \in dom(rule)$. Then due to definition 2 $\langle rule, X, N\rangle \wedge B \vdash_{RDR} c$ iff (b_1) $rule \wedge B \vdash c$ and $X \wedge B \not\vdash_{RDR} \bar{c}$ or (b_2) $X \wedge B \vdash_{RDR} c$. Case ($b_1$) does not occur if the α-rule is applicable since proposition 3 ensures that the scope is empty, i.e. there is no sample in the context of the node (= no sample that reaches this node when being classified) and in the domain of this node (the rule fires) that is not in the domain of any exception (see definition 5. In case (b_2) the result is the same as obtained by the right hand side RDR.*

The next partial equation, called β-rule reorders nodes, such that lists of succeeding nodes are turned into exceptions. By explicitly stating that rules are exceptions of other rules, the RDRs become more comprehensible. Additionally, in combination with the γ-rule, the number of literals in the RDR may be reduced.

Proposition 4 (β-rule)



iff $dom(rule) \subseteq dom(rule_N)$

Proof 4 *Let $c \in O$ be any literal. Let R be the left hand side and R' the right hand side RDR of proposition 4. (a) $c \in dom(rule)$. Then $R \wedge B \vdash_{RDR} c$ iff (a_1) $rule \wedge B \vdash c$ and $X \wedge B \not\vdash_{RDR} \bar{c}$ or (a_2) $X \wedge B \vdash_{RDR} c$. In case (a) proposition 4 ensures that $c \in dom(rule_N)$, hence $R' \wedge B \vdash_{RDR} c \Leftrightarrow \langle rule, X, N_X\rangle \wedge B \vdash_{RDR} c \Leftrightarrow rule \wedge B \vdash c$ (case a_1) or $X \wedge B \vdash RDRc$ (case a_2). (b) $c \notin dom(rule)$. We then have to distinguish two cases: (b_1) $c \notin dom(rule_N)$. Then $R \wedge B \vdash_{RDR} c$ iff $N \wedge B \vdash_{RDR} c$. In case ($b_1$) $R' \wedge B \vdash_{RDR} c$ iff $N \wedge B \vdash_{RDR} c$ as well. (b_2) $c \in dom(rule_N)$. Then $R \wedge B \vdash_{RDR} c$ iff $rule_N \wedge B \vdash c$ and $N_X \wedge B \not\vdash_{RDR} \bar{c}$ (b_{21}) or $N_X \wedge B \vdash_{RDR} c$ (b_{22}). In case (b_{21}) $R' \wedge B \vdash_{RDR} c$ because proposition 4 ensures $c \notin dom(rule)$, c was assumed to lie in $dom(rule_N)$ (b_{21}) and $N_X \wedge B \not\vdash_{RDR} \bar{c}$ (b_{21}). In case (b_{22}) $R' \wedge B \vdash_{RDR} c$ because again proposition 4 ensures $c \notin dom(rule)$.*

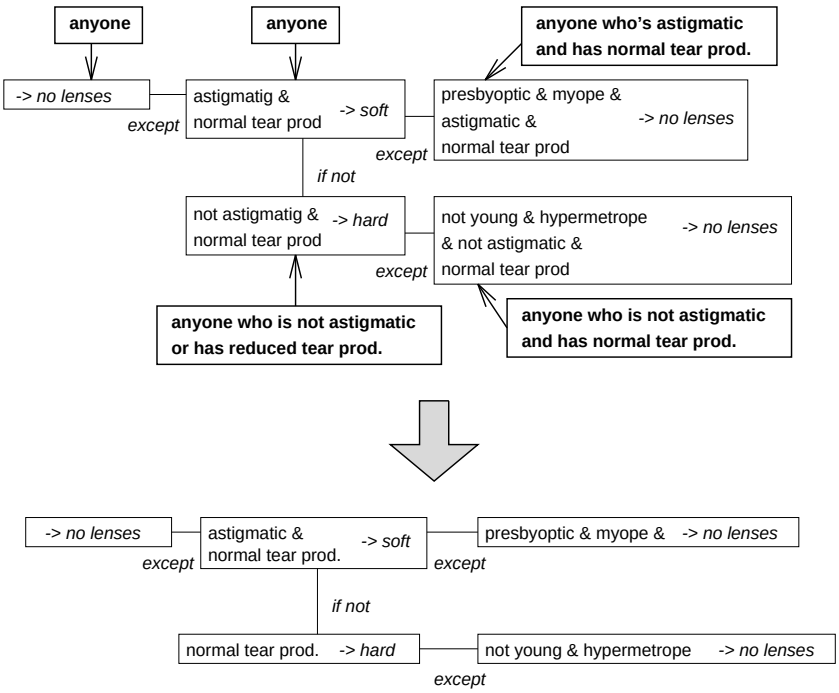


Figure 3: Example for the γ-rule

2.8 Reduction of propositional RDR

In this section, a third partial equation is presented, that motivates an efficient reduction algorithm for propositional rules as described in section 2.2. In case rules are horn clauses and the $\vdash$ relation is θ-subsumption, this rule becomes similar to the problem of θ-reduction, which cannot be solved efficiently.

Proposition 5 (γ-rule) $\langle head \leftarrow l_1 \wedge \ldots \wedge l_n, X, N\rangle \quad \equiv^{\gamma}_{O} \quad \langle head \leftarrow l_1 \wedge \ldots \wedge l_{n-1}, X, N\rangle$ *iff* $dom(head \leftarrow l_1 \wedge \ldots \wedge l_n) \cap con_O = dom(head \leftarrow l_1 \wedge \ldots \wedge l_{n-1}) \cap con_O$, *where* con_O *is referring to the node at which the rule is applied.*

Proof 5 *by induction over the structure of the tree* $\langle rule, X, N\rangle$ *The proposition trivially holds for the empty tree. Due to def. 2 for an arbitrary tree the exception branch may only affect the result if* $rule \wedge B \vdash \bar{c}$. *This implies* $c \in dom(rule)$ *and w.r.t def. 4* $x \in con_O$ *of* X. *The next tree may only influence the result if* $rule \wedge B \nvdash \bar{c}$ *(def. 2). Together with def. 4 this implies* $c \in con_O$ *of* N. *Thus, if the proposition holds for the subtrees* X *and* N, *it holds for* $\langle rule, X, N\rangle$.

Fig. 3 shows an example reduction. Each node's context is augmented boldly. Tests that are always satisfied within their context, are removed.

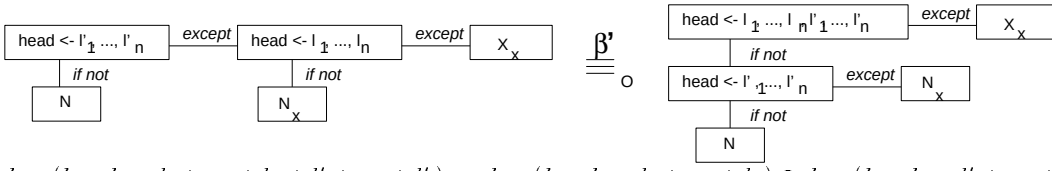
2.9 RDRs and decision lists

Decision lists [Riv87] are "flat" lists of rules which are applied only if none of their predecessors are applicable. That is, the first applicable rule classifies the input. Obviously, a RDR containing no exceptions is a decision list. In the following it is shown, that a RDR can be transformed into a decision list containing *at least* as many tests and each decision list can be transformed into a RDR containing *at most* as many tests.

An exception overshadows its next general rule. Removed and placed in front of its parent rule, it still overshadows the general rule but now it might fire although the parent rule does not

fire, therefore the parent rule's condition has to be conjunctively appended to obtain an equivalent rule:

Proposition 6 (β'-rule)



iff $dom(head \leftarrow l_1 \wedge \ldots \wedge l_n \wedge l'_1 \wedge \ldots \wedge l'_n) = dom(head \leftarrow l_1 \wedge \ldots \wedge l_n) \cap dom(head \leftarrow l'_1 \wedge \ldots \wedge l'_n)$

The condition of proposition 6 is static, due to proposition 1 it always holds for propositional languages. Hence, the equation can always be applied and the exception depth can be reduced until the RDR becomes a list of succeeding rules and all exception branches only contain the empty node λ.

Proof 6 *Let R be the left and R' be the right hand side RDR in proposition 6. Let $c \in O$ be any literal. (a) $c \in dom(head \leftarrow l'_1 \wedge \ldots \wedge l'_n)$ Due to definition 2, $R \wedge B \vdash_{RDR} c$ iff $head \leftarrow l_1 \wedge \ldots \wedge l_n \wedge B \vdash c$ and $\langle head \leftarrow l'_1 \wedge \ldots \wedge l'_n, X_X, N_X \rangle \wedge B \not\vdash_{RDR} \bar{c}$ (a_1) or $\langle head \leftarrow l'_1 \wedge \ldots \wedge l'_n, X_X, N_X \rangle \wedge B \vdash_{RDR} c$ (a_2). In case (a_1), $\langle head \leftarrow l'_1 \wedge \ldots \wedge l'_n, X_X, N_X \rangle \wedge B \not\vdash_{RDR} \bar{c}$ implies $c \notin dom(head \leftarrow l'_1 \wedge \ldots \wedge l'_n)$, which together with the condition of proposition 6 implies $c \notin dom(head \leftarrow l_1 \wedge \ldots \wedge l_n \wedge l'_1 \wedge \ldots \wedge l'_n)$, hence the topmost rule of R' does not fire, and $N_X \wedge B \not\vdash_{RDR} \bar{c}$, hence N_X does not fire. Therefore, $R' \wedge B \vdash_{RDR} c$. In case ($a_2$) we can distinguish two cases. (a_{21}) $c \in dom(head \leftarrow l_1 \wedge \ldots \wedge l_n)$, then due to the condition 6 $c \in dom(head \leftarrow l_1 \wedge \ldots \wedge l_n \wedge l'_1 \wedge \ldots \wedge l'_n)$ and $\langle head \leftarrow l_1 \wedge \ldots \wedge l_n, X, \ldots \rangle$ (in R) yields the same result as $\langle head \leftarrow l_1 \wedge \ldots \wedge l_n \wedge l'_1 \wedge \ldots \wedge l'_n, X_X, \ldots \rangle$ (in R'), and (a_{22}), $c \notin dom(head \leftarrow l_1 \wedge \ldots \wedge l_n)$. Then identical subtrees N_X fire in both, R and R'. (b) $c \notin dom(head \leftarrow l'_1 \wedge \ldots \wedge l'_n)$. Due to the condition of proposition 6 this implies $c \notin dom(head \leftarrow l_1 \wedge \ldots \wedge l_n \wedge l'_1 \wedge \ldots \wedge l'_n)$, hence in both RDRs, R and R', identical subtrees N fire and yield identical results.*

By repeatedly applying this rule, all exceptions can be removed. Note that by appending literals l'_i the description length is increased although some of the appended conditions may be removed afterwards as stated in proposition 5.

On the other hand, each decision list *is* a RDR (without exceptions). This trivially implies that a RDR has at most the same description length as a decision list. If the condition $dom(rule) \subseteq dom(rule_N)$ is satisfied for any two succeeding rules, proposition 4 (β-rule) may be applied and an exception may be generated. This may cause the reduction rule 5 (γ) to become applicable, hence a decision list can always be transformed into a RDR containing *at most* as many tests.

Fig. 4 shows how a RDR is transformed into a decision list by repeatedly placing exceptions in front of the next general rule. Note that the rule "young $\rightarrow$ don't" within the context of "birds $\rightarrow$ fly" has to be transformed to "young $\wedge$ birds $\rightarrow$ don't".

2.10 Ripple down rule sets

Since in a RDR each rule is implicitly augmented with the negations of all preceding rules, for better comprehensibility it is of interest to transform them into ripple down rule sets. This is always possible, but it requires, that the rule language includes both, conjunction of conditions and the *negation as failure*. $head(r)$ and $body(r)$ refer to the conclusion and the condition part of the rule respectively. If r_1 and r_2 are rules, then $body(r_1) \wedge not(body(r_2)) \rightarrow head(r_1)$ has to be in the rule language as well and $r \wedge B \vdash c \Leftrightarrow not(body(r)) \rightarrow head(r) \wedge B \not\vdash c$. The transformation of RDR-sets into RDRs is not possible in general, because while only one rule in a RDR may fire for a given observation or substitution, multiple rules may fire in a RDR-set. The following transformation augments a rule with the negation of a preceding rule. Applying this rule repeatedly, a RDR is transformed into a congruent RDR, such that only one rule in a list of successors fires; hence the RDR becomes an equivalent RDR-set.

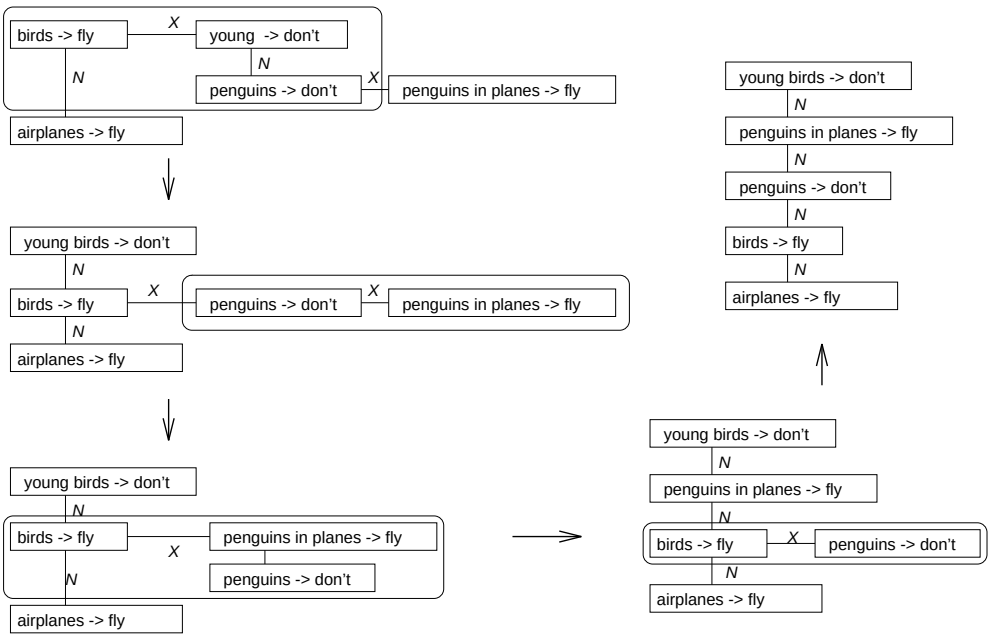


Figure 4: RDRs and decision lists

Proposition 7 (δ-rule) *Rules may be augmented with the negations of preceding rules:*

$$\langle r_1, X_1, \langle r_2, X_2, N\rangle\rangle \equiv^{\delta} \langle r_1, X_1, \langle body(r_2) \wedge not(body(r_1)) \rightarrow head(r_2), X_2, N\rangle\rangle \qquad (3)$$

Proof 7 *Let B be any background theory. Then due to definition 2 $\langle r_1, X, \langle body(r_2) \wedge not(body(r_1)) \rightarrow head(r_2), X_2, N\rangle\rangle \wedge B \vdash_{RDR} c$ iff $r_1 \wedge B \vdash c$ and $X \wedge B \nvdash_{RDR} \bar{c}$ (a), where $X = \langle r_2, X_2, N_2\rangle$, or $X \wedge B \vdash_{RDR} c$ and $r_1 \wedge B \vdash \bar{c}$ (b) or $N \wedge B \vdash_{RDR} c$ and $r_1 \wedge B \nvdash \bar{c}$ (c). Cases (a) and (c) are identical to the semantics of the RDR $\langle r_1, X_1, \langle r_2, X_2, N\rangle\rangle$. Now case (b) needs to be expanded further: $X \wedge B \vdash_{RDR} c$ and $r_1 \wedge B \vdash \bar{c}$ iff $body(r_2) \wedge not(body(r_1)) \rightarrow head(r_2) \wedge B \vdash c$ and $X_2 \wedge B \nvdash_{RDR} \bar{c}$ (b_a)) or $X_2 \wedge B \vdash_{RDR} c$ and $body(r_2) \wedge not(body(r_1)) \rightarrow head(r_2) \wedge B \vdash \bar{c}$ (b_b) or $N \wedge B \vdash_{RDR} c$ and $body(r_2) \wedge not(body(r_1)) \rightarrow head(r_2) \wedge B \nvdash \bar{c}$ (b_c), where $r_1 \wedge B \vdash \bar{c}$, which implies $r_1 \wedge B \vdash not(r_1)$ due to the definition of not. Hence, the equations can be simplified: $body(r_2) \rightarrow head(r_2) \wedge B \vdash c$ and $X_2 \wedge B \nvdash_{RDR} \bar{c}$ (b'_a) or $X_2 \wedge B \vdash_{RDR} c$ and $body(r_2) \rightarrow head(r_2) \wedge B \vdash \bar{c}$ (b'_b) or $N \wedge B \vdash_{RDR} c$ and $body(r_2) \rightarrow head(r_2) \wedge B \nvdash \bar{c}$ (b'_c). W.r.t the definition of head and body and definition 2, this is equivalent to $\langle r_2, X_2, N\rangle \wedge B \vdash_{RDR} c$, which is the (b) case of the left hand side of our equation.*

By repeatedly applying this transformation, a state is reached in which each rule in a list of successors contains the negations (negation as failure) of all preceding nodes. Hence only one of these rules fires and the semantics of ripple down rule sets yields the same set of conclusions as the RDR semantics. Figure 5 shows an example transformation.

3 Learning rules with exceptions

In this section, known approaches to induction of rules with exception are summarized and compared. A new algorithm with interesting properties is presented. The algorithms are described from a search in the hypothesis space point of view [Mit82]. The hypothesis space is a graph, the nodes of which are rules and the edges of which are set up by a more-specific-than relation. A rule r_1 is more specific than a rule r_2, if the instances covered by r_1 are a subset of the instances covered by r_2.

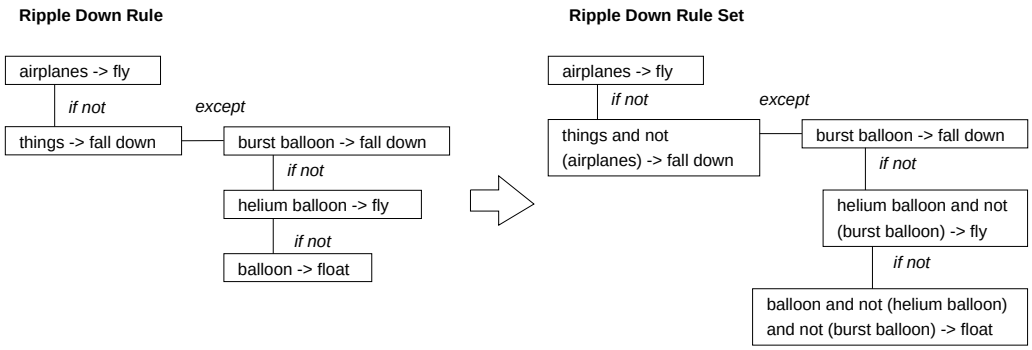


Figure 5: RDRs and RDR-sets

3.1 Learning as movement in the hypothesis space

Learning algorithms that generate a concept description by specializing an initially most general hypothesis, e.g. decision tree algorithms (e.g. [UW72, Qui86]), algorithms for induction of rules (e.g. [MMHL86]) or specialization based ILP techniques, e.g. MIS [Sha83], FOIL [Qui90], RDT [WK92], can be characterized by top-down movement starting from the topmost or a node close to the topmost node.

In a decision tree, each node is associated to a node in the graph that is given by the attributes/propositions tested so far. The insertion of a new test into a decision tree performed by a learning algorithm, corresponds to forking in the hypothesis graph by omitting one node and adding some of its successors, each of them corresponds to one of the newly generated leafs. Specializing an existing rule, as performed by rule induction algorithms accords to exchanging a node by one of its successors; setting up a new rule means to add a new node to the hypothesis. Thus, for these algorithms, movement in the hypothesis graph is top-down.

Learning by generalizing samples has mainly been performed in the area of ILP (e.g. [Plo70, MB88]). Generating the lgg means to exchange two nodes of the hypothesis graph by one nodes that has an edge to the original nodes, such that there is no other node with this property that has got an edge *from* this node (*least* general generalization), hence, bottom-up movement in the hypothesis graph.

Both, the specialization and generalization approach, can be combined into a bidirectional approach: An algorithm can use generalization (bottom-up) to find interesting rules and accept inexact rules if the rule fulfills some quality criterion. The rule can then be re-specialized (top-down) by appending a literal "not(ExceptionConcept)" and solving the nested learning problem by generalizing the counterexamples and thus generating rules for the newly invented exception concept. This principle is reflected in Vere [Ver80] and the newly proposed approach but not in [GC92], where learning is strictly top-down, or in [KMU93b], were learning is performed by searching a special region of the graph that is determined by a fixed exception depth.

3.2 Known approaches for induction of RDRs

Vere [Ver80] proposed the following algorithm *Counterfactuals* that contains the basic principle of most learning algorithms dealing with exceptions:

---

1. find an interesting rule by generalizing samples, that possibly covers some negative examples
2. learn the exceptions recursively (except branch)
3. learn the remaining samples recursively (if-not branch)

Gaines [GC92] proposed the similar algorithm *Induct* that is much more concrete in what is an interesting rule. It is obvious, that the quality of a RDR learning algorithm is strongly related to the strategy for finding interesting rules.

1. Given a conclusion, find the rule that is least likely to predict that conclusion by chance
2. learn the exceptions recursively (except branch)
3. learn the remaining samples recursively (if-not branch)

In [HSW89] an algorithm is proposed, that learns nested exceptions by repeatedly covering positive and negative examples, yet the algorithm does not deal with disjunctions of rules in the exception levels; this simplifies the learning problem.

In [DK95] an algorithm is presented, that uses a classical rule learning algorithm to find interesting rules and recursively learns their exceptions:

1. Use a classical algorithm to set up some interesting rules
2. if the rules are not more special than the next-general rule, then do not use them but enumerate the exceptions as ground clauses instead
3. learn the exceptions recursively

Step 2 addresses an problem that has to be taken into account by each algorithm: The rule covering the exceptions has to be more special than the next-general rule, or else the algorithm will not terminate but generate an infinitely large tree. The algorithm proposed in [HSW89] does not take that into account, because only one rule rather than a disjunction of rules is learned at each exception level, which is a major limitation.

In [KMU93b] Mannila et al. present an Occam algorithm that learns sets of rules with exception to a fixed depth by repeatedly solving weighted set covering problems.

In [Wro88] Wrobel proposes an extension of RDT, that counts counterexamples and after exceeding a threshold recursively learns an exception predicate. In contrast, the *most general correct specialization* [BM90] is asymmetrical in a way that counterexamples are excluded from a rule but are not generalized to form an exception concept.

3.3 The Cut95 algorithm

The Cut95 algorithm sets up ripple down rules, where rules are conjunctions of interval constraints. We represent samples as pairs $(c, (x_1, \ldots, x_n))$ of class symbols and attribute vectors. The algorithm tries to generalize existing rules with new samples using an implication based generalization operator notated $+$. Since rules are interval constraints for all x_i, i.e. cuboids in the attribute space, the most specific generalization of a rule and a new sample point is the least cuboids that includes both, the previous cuboid and the new point, i.e. the intervals have to be extended to include the new point.

The Cut95 algorithm rewrites a given RDR and a new example, written $\langle \cdot, \cdot, \cdot \rangle \leftarrow_\beta \mathit{sample}$, yielding a new RDR. The induction process in general starts with an empty RDR (λ), but if an initial knowledge base exists, it can be used as starting point. A dominance threshold β specifies a lower bound for the success rate with respect to the training set. In order to avoid over-fitting effects, rules that achieve the requested accuracy and still have counter-examples, are not refined.

1. $\lambda \leftarrow_\beta (c, (x_1, \ldots, x_n)) = \langle c \leftarrow [x_1, x_1], \ldots, [x_n, x_n], \lambda, \lambda \rangle$ – set up a new RDR if the current RDR is empty
2. $\langle rule, X, N \rangle \leftarrow_\beta (c, p) =$
   (a) $\langle rule, X \leftarrow_\beta (c, p), N \rangle$, if the rule is applicable and (X already classifies p or (p is a counter-example of the rule and the accuracy threshold β is not exceeded)).
   (b) $\langle rule, X, N \rangle$, if the rule is applicable, X does not classify c and (p is a positive example of the rule or the accuracy threshold β is exceeded).
   (c) $\langle rule + p, X, N \rangle$ (generalize rule with new point), if the rule is not applicable, p is a positive examples, no other class exceeds the accuracy threshold β in the new scope and (the variance of the positive samples exceeds the variance of the negative samples by a factor of δ or the probability of the positive samples exceeds β in the new scope).
   (d) $\langle rule, X, N \leftarrow_\beta c \rangle$ otherwise.

The algorithm passes those objects, that are an exception of the current rule, recursively into the exception branch. It tries to generalize new positive examples with the current rule using a heuristic criterion to check, whether the negative samples can be described by exception rules with less description length than would be required to break up the rules into several small rules covering only the positive examples. If the positive examples have a larger variance than the negative ones (the positive samples are spread across the domain, while the negative ones are locally bound), than most likely it is easier to cover the negative ones with exception rules. Instead of this variance criterion, the probability can be used (generalize, if the positive samples have a much higher probability than the negative ones). Throughout our experiments, we used a variance factor δ of 1.3.

3.3.1 Properties of the algorithm

The algorithm tries to include new samples into one of the early rules. A sample is passed to a succeeding node only if a generalization is not useful. The topmost rules extend and overshadow the succeeding rules, which after being overshadowed can be removed. Thus, the resulting RDRs are simplified during the training.

The algorithm is inherently incremental. If the probability or variance criterion is used, the algorithm is still incremental in so far, that the learning process does not need to be restarted on arrival of new samples, but the algorithm requires a set of samples to compute the variance of any box. Such a set can be accumulated and can be restricted to a fixed number of samples.

The formation of new rules and exceptions does not stop until each rule achieves the desired exactness β. That is, after termination the resulting RDR is guaranteed to achieve an exactness of at least β, even for very small training sets, unless no such classifier exists (identical objects are assigned different classes). A rule that achieves the desired exactness is no longer corrected, thus over-learning effects are avoided.

The algorithm learns classifications in continuous domains.

3.3.2 Examples

Fig. 3 shows a RDR that prescribes contact lenses (hard, soft or none) [Cen87]. The RDR can be considered quite compact and comprehensible, the corresponding rule set extracted from a decision tree constructed by *ID3* [Qui86] needs 9 rules and 30 tests while the system can be modeled with a RDR of 4 rules and 7 tests. Induct with exceptions [GC92] yields a tree of 4 rules and 8 tests, the difference is a benefit of the proposed reduction rule. Both algorithms yield exact classifiers (hit rate 1).

On the Shuttle dataset [MST94], containing 58,000 samples with discrete attributes and seven multimodally distributed classes, Cut95 outperforms all known machine learning, statistical and

neural network learning algorithms. Learning time did not exceed 15 minutes. Good results were also achieved for the diabetes dataset (12-fold cross-validation) [MST94]; although the dataset is very noisy and good hypotheses are very simple (DIPOL [SW94] only uses one hyper-plane), Cut95 outperforms all symbolic algorithms.

The experiment settings for both, the shuttle and the diabetes dataset were due to the specifications used in the ESPRIT project StatLog, described in [MST94]. A single trial was used for the shuttle dataset, while 12-fold cross-validation was performed to obtain a confident estimation of the error for the diabetes dataset. The training and test sets are disjoint, the following tabulars refer to the error for the unknown test sets.

contact lens prescription

algorithm	# rules	# tests
ID3	9	30
Induct	9	25
Induct exc.	4	8
Cut95	**4**	**7**

Diabetes

algorithm	test error	rank
Logdisc	0.223	1
Dipol92	0.224	2
Cut95	0.225	3
Discrim	0.225	3
SMART	0.232	5
Cal5	0.250	8
C4.5	0.270	13
NewID	0.289	19

Shuttle

algorithm	test error	rank
Cut95	**0.001**	1
NewId	0.01	2
Baytree	0.02	3
Cal5	0.03	4
CN2	0.03	4
CART	0.08	5
C4.5	0.1	7
AC^2	0.32	9
Backprop	0.43	10
kNN	0.44	11
RBF	1.4	16
Discrim	4.83	20

4 Conclusion and Further Research

Ripple down rules prove to be an interesting and powerful scheme for knowledge representation. They achieve a significant increase of comprehensibility and compactness of representation; they obtain a shorter description length than decision lists. The proposed semantics covers first order rule languages, the algebraic foundations provide transformation of RDRs into RDR-sets, decision lists, and vice versa, as well as an algorithm for simplification of RDRs.

The proposed algorithm learns rules in continuously attribute-value based domains. Learning is performed by bottom-up generalization of samples and top-down specialization of exception, thus over-generalization can be fixed. The algorithm is guaranteed to yield a classifier of any desired exactness if such a classifier exists and yields good results for real-world classification tasks.

In our future work, we will use this algorithm to learn concepts from structural examples, using a first-order language for rules and structural generalization operator. We will further modify the algorithm in order to learn continuous functions.

ACKNOWLEDGMENT

This work was partially supported by an Ernst-von-Siemens Fellowship held by the author. Special thanks are due to Ralf Herbrich, who implemented some of the presented algorithms, to Paul Compton for helpful remarks and to Fritz Wysotzki for interesting discussions. Parts of this work were done when the author was at Siemens Corporate Research, Princeton.

References

[BM90] M. Bain and S. Muggleton. Non-monotonic learning. In J. E. Hayes-Michie and E. Tyugu, editors, *Machine Intelligence*, volume 12, pages 105–119, 1990.

[Cen87] J. Cendrowska. An algorithm for inducing modular rules. *International Journal for Man-Machine Studies*, pages 349–370, 1987.

[CJ88] P. Compton and R. Jansen. Knowledge in context: a strategy for expert system maintainance. In *Proceedings of the 2nd Australian Joint Artificial Intelligence Conference*, volume 406 of *LNAI*, pages 292–306, Adelaide, 1988. Springer.

[CPKY94] P. Compton, P. Preston, B. Kang, and T. Yip. Local patching produces compact knowledge bases. In *A Future for Knowledge Acquisition / EKAW '94*. Springer Verlag, 1994.

[DK95] Yannis Dimopoulos and Antonis Kakas. Learning non-monotonic logic programs: Learning exceptions. In Nada Lavrač and Stefan Wrobel, editors, *Machine Learning: ECML-95 (Proc. European Conf. on Machine Learning, 1995)*, Lecture Notes in Artificial Intelligence 912, pages 122 – 137. Springer Verlag, 1995.

[GC92] B. R. Gaines and P. J. Compton. Induction of ripple down rules. *5th Australian Conference on Artificial Intelligence*, pages 349–355, 1992.

[HSW89] D. Helmbold, R. Sloan, and M. K. Warmuth. Learning nested differences of intersection-closed concept classes. *Proceedings of the Workshop on Computational Learning Theory*, pages 41–56, 1989.

[KCP95] B. Kang, P. Compton, and P. Preston. Multiple classification ripple down rules: evaluation and possibilities. In B. Gaines and M. Musen, editors, *Proc. 9th Banff Knowledge Acquisition for Knowledge Based Systems Workshop*, pages 17.1–17.20, 1995.

[KMU93a] J. Kivinen, H. Mannila, and E. Ukkonen. Learning rules with local exceptions. In *Proceedings of the European Conference on Computational Learning Theory (COLT)*, 1993.

[KMU93b] Jyrki Kivinen, Heikki Mannila, and Esko Ukkonen. Learning hierarchical rule sets. *Proceedings of the ACM Workshop of Computational Learning Theory*, 1993.

[KNS92] B. Kijsirikul, M. Numao, and M. Shimura. Discrimination-based constructive induction of logic programs. In *Proc. National Conf. on AI (AAAI)*, pages 44–49, 1992.

[MB88] S. H. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In *Proc. 5th International Conference on Machine Learning*, pages 339–352, 1988.

[Mit82] T. M. Mitchell. Generalization as search. *AI Journal*, 18:203–226, 1982.

[MMHL86] R. S Michalski, I. Mozetic, J. Hong, and N Lavrac. The multi-purpose incremental learning system aq15 and its testing application to three medical domains. *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 1041–1045, 1986.

[MST94] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[PEC94] P. Preston, G. Edwards, and P. Compton. A 2000 rule expert system without a knowledge engineer. In *Proc. AAAI Knowledge Acquisition for Knowledge Based Systems Workshop*, 1994.

[Plo70] G. D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 153–163, 1970.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1986.

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

[Riv87] R. L. Rivest. Learning decision lists. *Machine Learning*, 2(2):229–246, 1987.

[Sha83] E. Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, 1983.

[SW94] B. Schulmeister and F. Wysotzki. The piecewise linear classifier dipol92. In F. Bergadano and L. De Raedt, editors, *Machine Learning: ECML-94*, LNAI 784. Springer Verlag, 1994.

[UW72] P. Ulbrich and F. Wysotzki. Metaalgoritmen zur konstruktion von klassifizierungsalgorithmen und ihre simulation auf rechenautomaten. *Kybernetik-Forschung*, 1, 1972.

[Ver80] S. A. Vere. Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence*, 14:139–164, 1980.

[WK92] S. Wrobel and J.-U. Kietz. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming*, London, 1992. Academic Press.

[Wro88] S. Wrobel. Automatic representation adjustment in an observational discovery system. In *Proc. European Working Session on Learning*, 1988.