Reports of Machine Learning and Inference Laboratory, MLI 94-7, Center for MLI, George Mason Univ., December 1994.

CONCEPTUAL TRANSITION FROM LOGIC TO ARITHMETIC IN CONCEPT LEARNING

Janusz Wnek and Ryszard S. Michalski

Center for Machine Learning and Inference
George Mason University
4400 University Dr.
Fairfax, VA 22030

Abstract

This paper presents a computational study of the change of the logic-based concepts to arithmetic-based concepts in inductive learning from examples. Specifically, we address the problem of learning concepts whose description in the original knowledge representation space is very complex and difficult to learn using a conventional machine learning approach. By detecting "exclusive-or" patterns in the initially created hypotheses, the system postulates symmetry relations among pairs of initial attributes. The symmetry pairs are then combined into maximal symmetry groups. Each symmetry group leads to a creation of a *counting attribute* which is added as a new dimension to the representation space. So modified representation space facilitates the determination of new type of concept descriptions, called *conditional M-of-N rules.* The proposed ideas are illustrated by a visualization method based on the generalized logic diagram (GLD).

Keywords: constructive induction, representation change, concept learning

ACKNOWLEDGMENTS

The authors thank Ken Kaufman for his comments on this work. This research was conducted in the Center for Machine Learning and Inference at George Mason University. The Center's research is supported in part by the National Science Foundation under Grants No. CDA-9309725, IRI-9020266, and DMI-9496192, in part by the Advanced Research Projects Agency under Grant No. N00014-91-J-1854, administered by the Office of Naval Research, the Grant No. F49620-92-J-0549, administered by the Air Force Office of Scientific Research, and in part by the Office of Naval Research under Grant No. N00014-91-J-1351.

1 INTRODUCTION

Traditional concept learning methods express the learned hypothesis using attributes or terms that are present in the training examples. In other words, they learn in the same representation space in which training examples are presented. For many practical problems this is a serious limitation, because concepts to be learned require attributes or terms that go beyond those originally provided.

To attack such problems, a constructive induction approach splits the learning process into two intertwined searches: one–for the most appropriate representation space for the given learning problem, and second—for the "best" inductive hypothesis in the newly created space (Wnek and Michalski, 1994b).

The search for the most appropriate representation space is performed by applying *constructive induction* operators ("CI operators") that perform problem-oriented modifications of the representation space. Such a process can involve three types of space transformations: generating new, more relevant dimensions (inventing new descriptive concepts), removing less relevant dimensions (representation space reduction or "feature selection") and changing the quantization level of dimensions (dimension abstraction). There are many specific operators for performing such changes (especially of the first type). Searching through possible representation spaces is, therefore, impractical without some powerful shortcuts. Thus, the fundamental research problem in constructive induction concerns ways of reducing this search by inventing heuristics and methods that govern the search for the best representation.

This paper addresses a class of learning problems for which conventional symbolic methods produce DNF-type descriptions that are prohibitively long. The essence of the problem is that these methods cannot simply represent concepts that involve counting of properties detected in an object. An example of a "counting property" is the M-of-N concept (at least M out of N properties of a certain kind are present in an object). Problems of this type occur in various real-world problems, for example, in medicine (Spackman, 1988), planning (Callan & Utgoff, 1991), game playing (Fawcett & Utgoff, 1991), biology (Baffes & Mooney, 1993) and biochemistry (Towell & Shavlik, 1994).

The proposed method addresses a class of problems that require learning descriptions combining one or more M-of-N concepts with one or more DNF expressions. Such combined descriptions are called *conditional M-of-N* rules. The well-known M-of-N rules are thus a special case of conditional M-of-N rules.

The representational mechanism proposed extends the meaning of M-of-N concepts. The usual meaning of M-of-N is that "at least M out of N possible properties are present in an object." In the presented method, an arbitrary subset of "M" values can be easily represented, which facilitates learning such concepts as “parity” (even or odd number of counts) or an arbitrary count of any properties occurring in an entity. The method thus significantly extends the class of concepts that can be easily learned and represented.

2. RELATED RESEARCH

The proposed method is closely related to the earlier work by Michalski (1969) and Jensen (1975) on the detection of symmetry in logic functions, and the introduction of the *counting attribute* inductive generalization rule (Michalski, 1983). It is also related to the work by Seshu (1989) on learning concept descriptions that are logically equivalent to a combination of DNF and M-of-N rules, but in the context of decision tree learning. His method determines "best" XOR combinations in a randomly selected subset of the original attributes. Although this method improves the prediction accuracy of learned descriptions, the attributes created this way have unclear meaning. Also, due to the randomness of selection, the system may miss important relationships.

Our recent research in this direction was triggered by the difficulties experienced by well-known symbolic learning systems in solving the MONK2 problem (one of three problems employed in the international competition of learning programs, organized on the occasion of the International

School in Machine Learning in Belgium, 1991) (Thrun et al, 1991; Wnek & Michalski, 1994a). The MONK2 problem is to learn the concept: "Exactly two of the six attributes have their first value," which is a special case of the M-of-N concept. Learning this concept turned to be very difficult for methods oriented toward learning DNF-type descriptions (a logical disjunction of conjunctive concepts), such as decision tree or decision rule learning methods. The reason for the difficulty is that a learning system needs to represent an M-of-N concept, which is very difficult using DNF expressions.

In recent years there have been several efforts on learning M-of-N concepts. Spackman (1998) developed the CRLS system that learns *M-of-N* rules by employing non-equivalence symmetry bias and criteria tables. Murphy & Pazzani (1991) developed the ID-2-of-3 system that incorporates *M-of-N* tests in decision trees. Bloedorn & Michalski (1991) developed AQ17-DCI a program that employs a variety of operators to construct new attributes. Fawcett and Utgoff (1991) used an attribute representation similar to the Michalski's (1983) counting arguments generalization rule to expand the original representation space. Callan and Utgoff (1991) developed a restricted form of the counting arguments rule to create a numeric function from a Boolean expression that begins with a universal quantifier. More recently, Baffes & Mooney (1993) introduced the NEITHER-M-of-N system, which refined M-of-N rules by increasing or decreasing either M or N.

The method described here aims at learning descriptions that combine M-of-N concepts with DNF expressions. To this end, it searches for attribute symmetry that is evidenced by "exclusive-or" (XOR) patterns in the initially constructed DNF concept descriptions. Based on the XOR-patterns the system postulates symmetry relations among pairs of initial attributes. The symmetrical attributes are combined into maximal symmetry classes. Each symmetry class leads to creation of a *counting attribute* which is added as a new dimension to the representation space. The new counting attributes enable representing and learning a class of learning problems that require combination of logic-type (DNF) and arithmetic-type (M-of-N) concepts, i.e., *conditional M-of-N rules*.

3 THE AQ-HCI METHOD

3.1. Patterns in Descriptions

The proposed AQ-HCI method is hypothesis-driven, which means that changes of the concept representation space are proposed by analyzing hypotheses generated in that space rather than by analyzing training data. The method is called AQ-HCI, because it combines an AQ-type inductive rule learning system (specifically, AQ15; Michalski et al., 1986) with a procedure for an iterative hypothesis-driven constructive induction (HCI) for transforming the representation space.

Changes in the representation space are based on detecting certain regularities, called *patterns*, in concept descriptions (rulesets). Patterns detected in descriptions obtained in one iteration are used to transform the representation space for the next iteration. In this context, a pattern is a component, or a set of components of a description that accounts for a significant number of training examples. The "significant" number is defined by a user-specified parameter. Our earlier work in this area involved search for three types of patterns: value-patterns, condition-patterns, and rule-patterns. Value-patterns are subsets of frequently co-occurring attribute values (they are aggregated into single, more abstract values). Condition-patterns represent a conjunction of two or more elementary conditions. A rule-pattern consists of a subset of rules. Detecting such patterns and using them for enhancing the representation space have proven to be highly effective in improving the performance accuracy in DNF-type learning problems (Wnek & Michalski, 1994b).

In addition of the above patterns, we have recently also introduced *functional-patterns*, which are certain subfunctions occurring in a class description. For example, symmetrical functions XOR (exclusive-or) or EQ (equivalence) of two (or more) attributes in a description may constitute a pattern if they cover a significant number of training examples. XOR patterns are particularly interesting, because they are directly relevant to determining M-of-N concepts. The XOR-patterns identify subsets of attributes related to the M-of-N arithmetic subconcept involved in the description of the learned concept and enable the arithmetic subconcept to emerge through the application of the counting attribute generation rule.

Sections 3.2-3.3 explore the problem of detecting symmetry in concept descriptions. Section 3.4 introduces the *counting attribute generation rule*. Section 3.5 justifies the rule by showing a multistage process of constructing counting attributes, and relates logic to arithmetic. Section 3.6 describes an application of the counting attribute generation rule in an algorithm for improving the original representation space. Section 4 presents a case study of learning the MONK2 problem. Section 5 gives results from experiments performed on various conditional M-of-N concepts.

3.2 Symmetrical Functions And Concepts

Concept learning from examples can be viewed as learning an incompletely specified discrete function (Michalski, 1969; Michalski, Rosenfeld and Aloimonos, 1994). The function is incompletely specified because the set of examples usually never exhausts all possible instances of that function. In single concept learning, the function takes value 1 for any instance that belongs to the concept and value 0, otherwise. In "multiple-valued" concept learning, the function takes more than two discrete values.

Following Michalski (1969), a function $f(x_1, x_2, ..., x_n)$ is called *symmetrical with respect to variables* $\{x_i, x_j\}$ if

$$f(..., x_i, ..., x_j, ...) = f(..., x_j, ..., x_i, ...) \qquad (1)$$

where, n >= 2, 0 < i <= n, 0 < j <= n, i <> j.

From now on, we assume for the sake of simplicity that the two symmetrical variables are placed as the first two arguments of the function, and the remaining variables are denoted by R ("rest"). Since concepts can be characterized as discrete functions, we introduce the following definition. A concept C(x, y, R) is called *symmetrical with respect to attributes x and y*, if for all instances of that concept:

$$C(x, y, R) = C(y, x, R) \qquad (2)$$

This definition implies that the domains (value sets) of the attributes x and y are the same. The definition is applicable for both single and multiple-valued concept learning. Symmetry of the concept C(x, y, R) with respect to attributes x and y implies existence of a symmetrical relation between instances of that concept. There are two interesting cases of symmetrical functions that can be embedded within a concept description.

(1) Equivalence symmetry — EQ_{xy}: (x & y or ~x & ~y)

Figure 1 illustrates the equivalence symmetry. The representation space is defined by 4 binary attributes, x0, x1, x2, x3. The figure shows projection of a concept on the area described by (x0 & x2) or (~x0 & ~x2). The described concept is characterized by values + (positive examples) and - (negative examples). The gray area in the diagram marks that part of the concept that is irrelevant to the equivalence symmetry of x0 and x2. Examples e1 and e2 are described by the following VL1 formulas (VL1 stands for the variable-valued logic system 1, which is a form of propositional calculus; Michalski, 1975):

e1 <:: **x0 & x2** & x1 & ~x3
e2 <:: ~**x0 & ~x2** & x1 & ~x3

Example e1 has two "1s" as values of x0 and x2. Example e2 has two "0s" as values of those attributes. Attribute values of the remaining attributes, x1 and x3, are the same for both examples. The name "equivalence symmetry" comes from the fact that both attributes take always the same value. The diagram to the right, has a different arrangement of attributes. Attributes x0 and x2 were switched. Concept examples have correspondingly switched places; however, concept values were not changed due to the equivalence symmetry. The areas that switched examples are described by the following VL1 formulas:

S1 <:: **x0 & x2**
S2 <:: ~**x0 & ~x2**

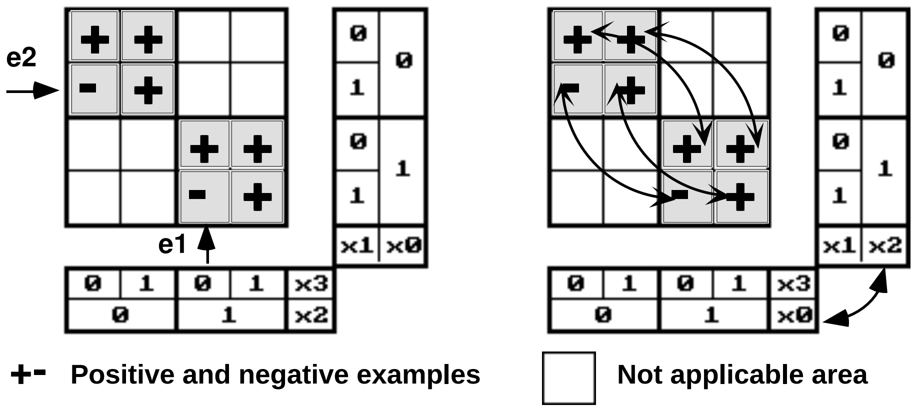


Figure 1. Equivalence symmetry with respect to attributes x0 and x2

(2) Exclusive-or symmetry — XOR$_{xy}$: (x & ~y or ~x & y)

Figure 2 illustrates the exclusive-or symmetry. The representation space is defined by 4 binary attributes, x0, x1, x2, x3. The figure shows a projection of three concepts (C1, C2, C3) onto the area described by (x0 & ~x2) or (x2 & ~x0). The gray area in the diagram marks the part of the concept that is irrelevant in the exclusive-or symmetry of x0 and x2. Examples e1 and e2 are described by the following VL1 formulas (VL1 stands for the variable-valued logic system 1, which is a form of propositional calculus; Michalski, 1975):

e1 <:: **~x0 & x2** & ~x1 & ~x3
e2 <:: **~x2 & x0** & ~x1 & ~x3

Example e1 has two "1s" as values of x0 and x2. Example e2 has two "0s" as values of those attributes. Attribute values of the remaining attributes, x1 and x3, are the same for both examples. The name "exclusive-or symmetry" comes from the fact that the attribute values form an exclusive-or. The diagram to the right has a different arrangement of attributes. Attributes x0 and x2 were switched. Concept examples have correspondingly switched places, however, concept values were not changed due to the exclusive-or symmetry. The areas that switched examples are described by the following VL1 formulas:

S1 <:: **~x0 & x2**
S2 <:: **~x2 & x0**

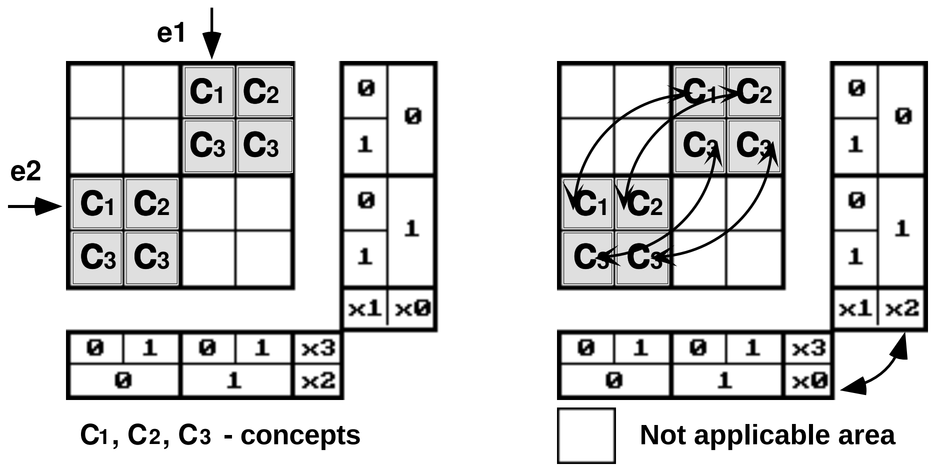
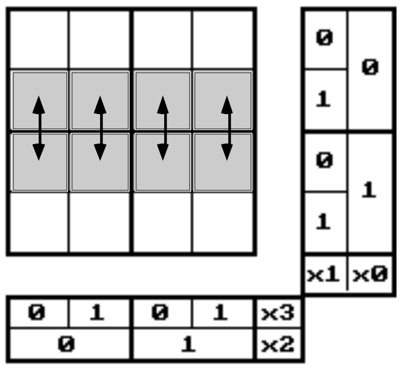
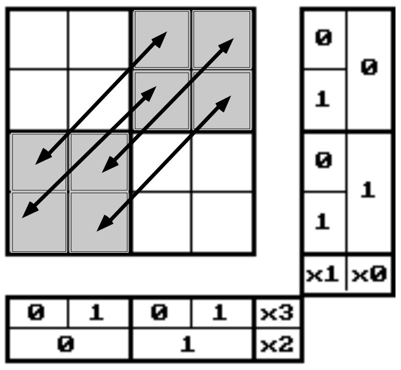


Figure 2. Exclusive-or symmetry with respect to attributes x0 and x2

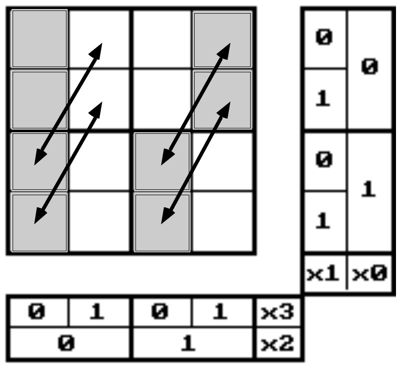
Figure 3 illustrates all possible XOR symmetries between the four attributes. Each symmetrical pair of examples must have the same class membership but different pairs can belong to different classes.



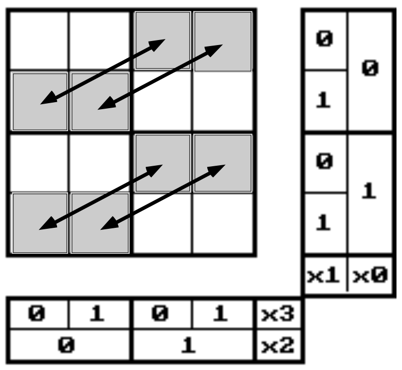
A. x0 XOR x1



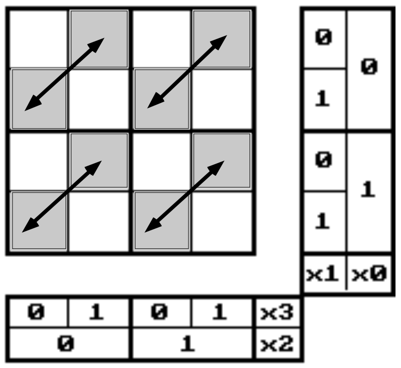
B. x0 XOR x2



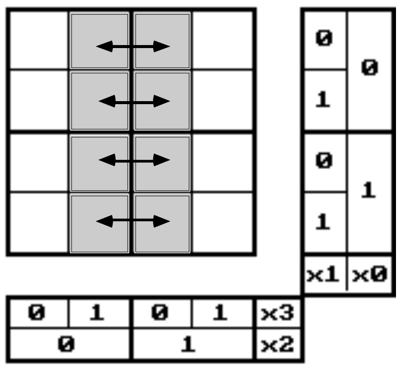
C. x0 XOR x3



D. x1 XOR x2



E. x1 XOR x3



F. x2 XOR x3

Figure 3. Visualization of all possible XOR symmetries between attributes x0,x1,x2,x3

An important property of the XOR symmetrical function is that it is transitive.

Theorem. The XOR relation is transitive.

$$\forall\ x, y, z\ XOR_{xy}\ \&\ XOR_{yz}\ \Rightarrow XOR_{xz} \qquad (3)$$

Proof.

We prove that for every example covered by XOR_{xz} there exists a symmetrical example, i.e. XOR(~x, y, z, R) = XOR(x, y, ~z, R).

XOR(~x, y, z, R) = XOR(x, ~y, z, R) { Based on XOR_{xy}}

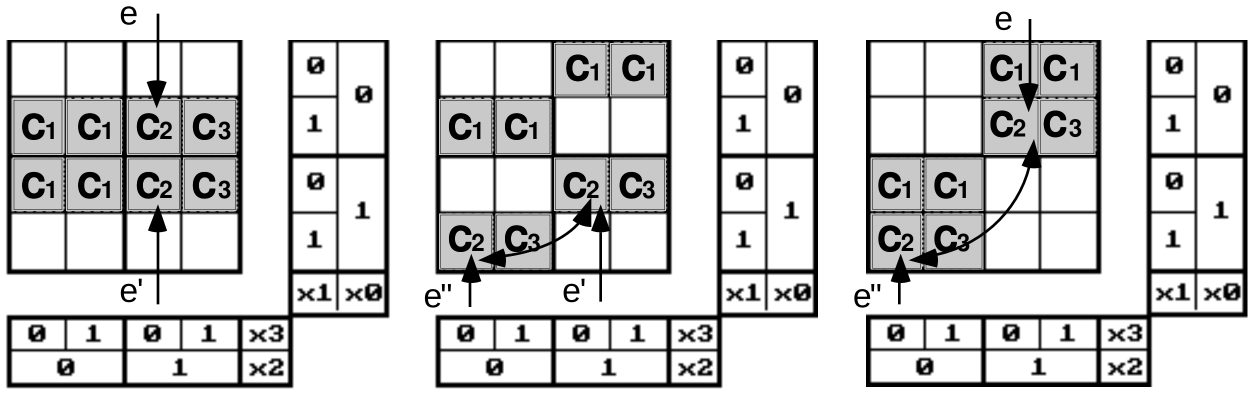
XOR(x, ~y, z, R) = XOR(x, y, ~z, R) { Based on XOR_{yz}} Q.E.D.

Figure 4 illustrates the transitivity property. From the fact that a concept is symmetrical with regard to two pairs of attributes (x0, x1) and (x1, x2), it is easy to see that the concept is also symmetrical with regard to the pair (x0, x2). A demonstration is given for the two examples of concept C2. In a similar way, symmetrical examples can be found for all other examples covered by the XOR(x0, x2, R) symmetry. Below, the two steps are also illustrated using VL1 descriptions of examples, e, e', e''.

(e ∈ C2) <:: **~x0 & x2** & x1 & ~x3

(e' ∈ C2) <:: **x0** & x2 & **~x1** & ~x3 ; XOR(x0,x1,R) applied

(e" ∈ C2) <:: x0 & **~x2** & **~x1** & ~x3 ; XOR(x1,x2,R) applied



XOR(x0, x1, R) **XOR(x1, x2, R)** **=>** **XOR(x0, x2, R)**

Figure 4. Transitivity of the XOR symmetrical relation

The following lemma and theorem link symmetrical concepts with M-of-N concepts.

Lemma

For all M-of-N concepts; for each pair of attributes (x,y); for each concept example (counterexample) with (x=1 & y=0) or (x=0 & y=1) values, there exists one and only one symmetrical example (counterexample) with (x=0 & y=1) or (x=1 & y=0) values, respectively.

Proof

Let e+ be a positive example of a M-of-N concept C. Let the (x,y) attribute pair have the following values (x=1 & y=0). Let Y be the set of counts. This set does not change if the (x,y) has values (x=0 & y=1) because the number of "0s" and "1s" remains the same. This means that e' created by switching values of the two attributes is also example of the concept C. There is only one such example because of the way e' is constructed. The proof for negative examples is similar. Q.E.D.

Theorem 2

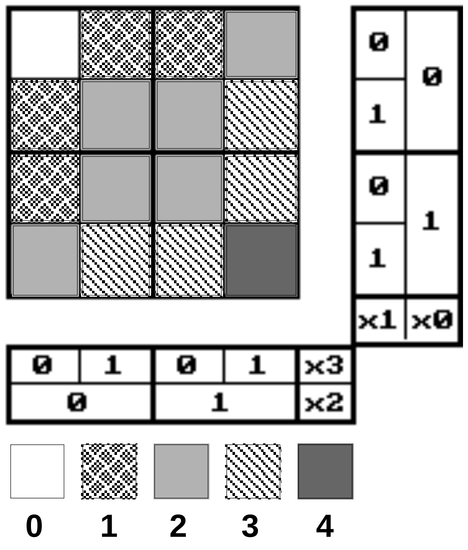
C is a M-of-N concept if and only if it is XOR symmetrical with respect to each pair of the N attributes.

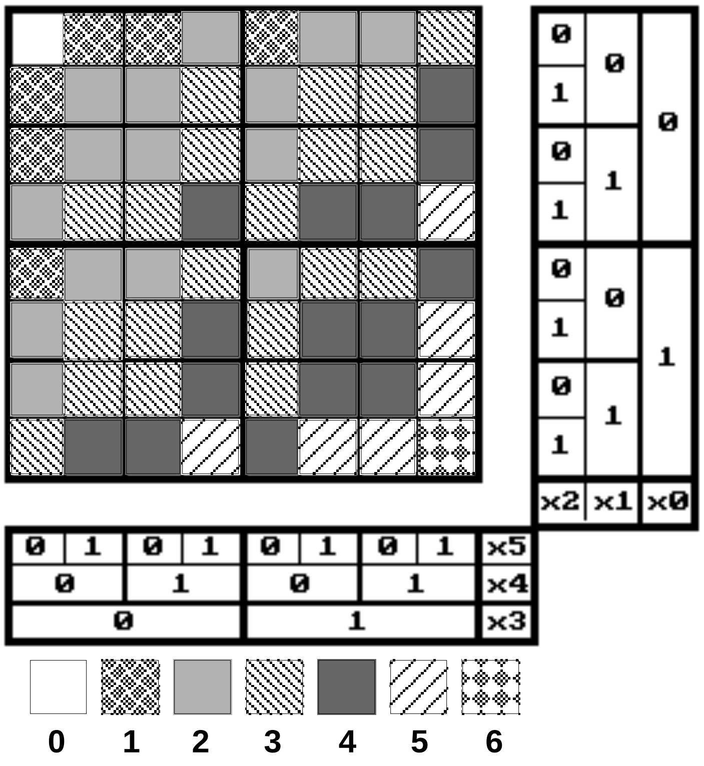
Proof.

Based on the lemma, the fact that C is M-of-N concept implies that for all pairs from N attributes the concept is XOR symmetrical.

Proving reverse implication is obvious. Q.E.D.

Figures 5A and 5B illustrate representation space in which all attributes are XOR symmetrical with each other. In Fig. 5A, there are five distinct symmetrical areas depicted by five different patterns. By assigning various class memberships to each pattern it is possible to create many different, symmetrical with respect to the four attributes, combinations of multiple-valued concepts. Since there may be up to five different concepts (values of concepts) represented and they can be assigned to five different areas, there are 4^5 (4096) possible symmetrical combinations. (This calculation is similar to using four-based counting system, on five positions). In the case of representing positive and negative examples of a single concept, there are 2^5 (32) possible combinations. Fig. 5B illustrates 6 dimensional symmetrical representation space. Fig. 6 shows the 4D representation space with a possible arrangement of five symmetrical concepts.

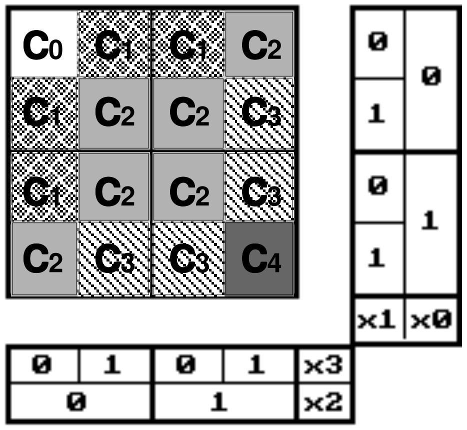




A. 4-dimensional representation space.

B. 6-dimensional representation space.

Figure 5. Symmetrical representation spaces.



C0, C1, C2, C3, C4 - concepts

Figure 6. Five symmetrical concepts.

Figure 7A shows the 4D representation space and a possible arrangement of two symmetrical, with respect to the four attributes, concepts C0 and C1. In such a binary setting, one concept can be regarded as a negation of the other one. Examples of the target concept are usually labeled as "positive" and its negation as "negative." This is visualized in the diagram in Fig. 7B. The concept labeled with "C1" or "+" is the "odd" concept. Its negation is the "even" concept. Both of them are cases of symmetrical (or M-of-N) concepts: odd-of-4 and even-of-4 (Figure 8).

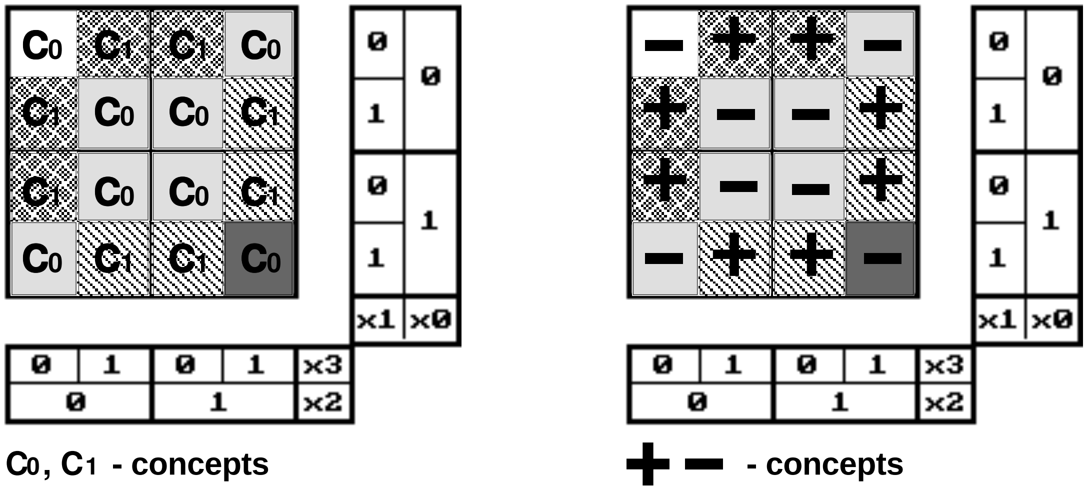
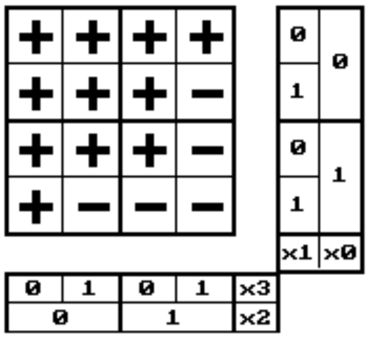
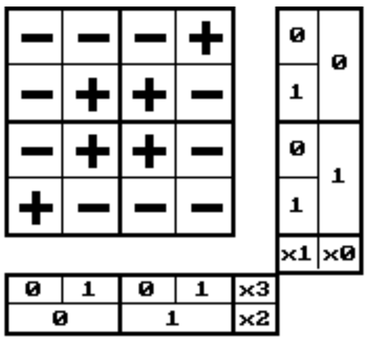
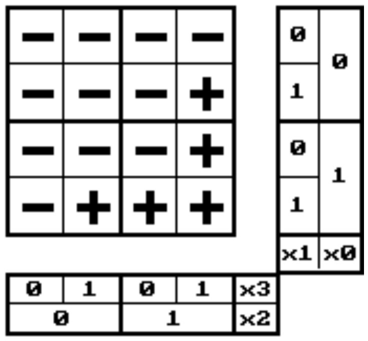
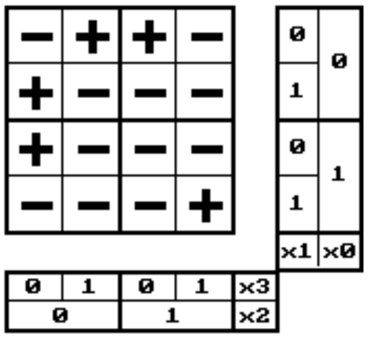


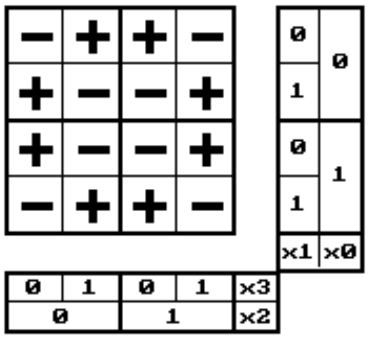
Figure 7. Two symmetrical parity concepts: odd (C1, +) and even (C0, -).


A. At most 2-of-4


B. Exactly 2-of-4


C. At least 3-of-4
(standard M-of-N concept)


D. 1 or 4 of-4


E. Odd-of-4

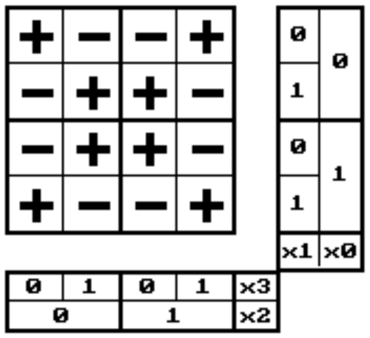

F. Even-of-4

Figure 8. Examples of symmetrical concepts in 4-dimensional representation space.

3.3 Relational Conditions and Symmetry Classes (SC)

Let Ci be relational conditions on values of single attributes, such as xi=5, xi>3 or xi=2..5 (as defined in variable-valued logic VL1, Michalski, 1975). Relational conditions evaluate to *true* (1) or *false* (0). Such conditions are building blocks in the rules generated by the rule learning system AQ15 employed in the proposed method (Michalski et al., 1986).

Let R1 and R2 be two conjunctive rules in a ruleset representing a hypothesis. Suppose R1 can be represented as Ci & ~Cj & CTX1 and R2 as Cj & ~Ci & CTX2, where CTX1 and CTX2 are "context" conditions, expressed in the form of a conjunction of zero or more relational conditions. It is said that Ci and Cj represent a "binary symmetry class" for the hypothesis, if CTX1 and CTX2 are in a *subsumption* relation, that is, CTX1 = CTX2 & CTX3 or CTX2 = CTX1 & CTX3, where CTX3 is a context condition.

Given binary symmetry classes (SCs) that involve k relational conditions, they can be combined into a k-ary symmetry class, if they have non empty intersections with each other. For example, if SC1 = {c1, c3}, SC2 = {c1, c2} then they can be combined into SC3 = {c1, c2, c3} due to the transitivity of the XOR relation.

A maximum symmetry class is a set with a maximal number of relational conditions that can be combined together.

3.4 The Counting Attribute Generation Rule

Suppose a k-ary maximum symmetry class of attributes (or relational conditions) {A1, A2, ...Ak} has been detected in a given concept description. The "counting attribute generation rule" (a constructive induction operator) suggests to create in such a situation a new attribute (dimension in the representation space), called the *counting attribute,* CA, defined by the arithmetic sum [A1+A2+...+Ak].

If the attributes are binary (or relational conditions evaluate to 0 or 1), the counting attribute sums up the "evidence" contributed by individual attributes or conditions in the maximum symmetry class. Values of the counting attribute represent the number of relational conditions (binary attributes) that hold for the given concept example, and its domain (value set) is the set of integer values from 0 to k. If the conditions use multivalued attributes then the counting attribute sums up the values of constituent attributes. The following notation is introduced for representing a counting attribute:

#AttrIn{Attribute Set: IREL VAL} (4)

where, *Attribute Set* is a list of attributes (from the maximum symmetry class in our method), *IREL* specifies an "internal" relation EQ, NEQ, GT, LT, GE or LE and *VAL* specifies a value. Such an attribute is read: "The number of attributes in the Attribute Set is in relation IREL with VAL." If attributes are binary, then the notation for the counting attributes can be simplified to: #AttrIn{Attribute Set}. To express a relational condition using a counting attribute, one writes:

#AttrIn{Attribute Set : IREL VAL} REL Values (5)

where REL is one of {=, <>, <, >, >= or <=}, and Values stands for one or more possible values of the attribute linked by the internal disjunction or the range operator ("..").

For example, to express M-of-N concept ("At least M out of N properties from {P1, P2, ...PN} hold" one would write: #AttrIn{ P1, P2, ...PN} ≥ M. To express the condition "Between 2 and 4 (non-binary) attributes in the set {A2, A3, A5, A7, A9, A12} have value greater than 5, one would write: #AttrIn{A2, A3, A5, A7, A9, A12: GT 5} = 2 v 4.

To illustrate the advantage of introducing the counting attribute, let us describe the concept represented in Fig. 9C. The simplest DNF description of this concept is:

~x0 & ~x1 & ~x2 & ~x3 or
~x0 & ~x1 & x2 & x3 or
~x0 & x1 & ~x2 & x3 or
~x0 & x1 & x2 & ~x3 or
x0 & ~x1 & ~x2 & x3 or
x0 & ~x1 & x2 & ~x3 or
x0 & x1 & ~x2 & ~x3 or
x0 & x1 & x2 & x3

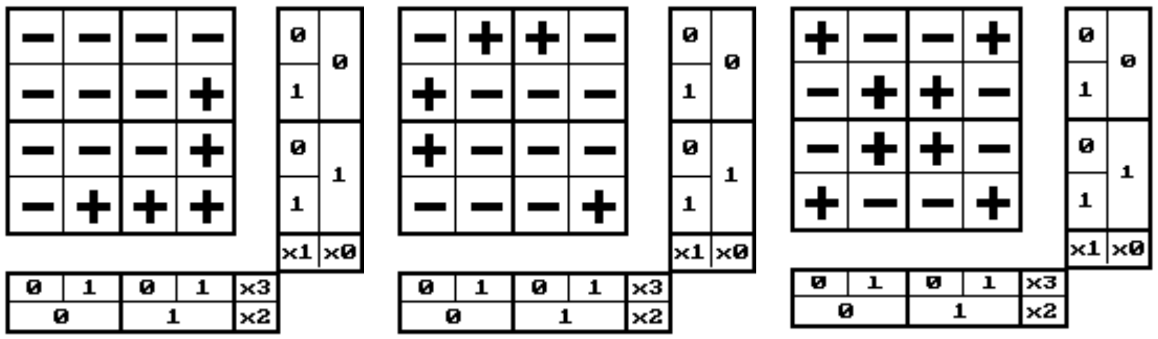
The logically equivalent expression using the counting attribute is

[#AttrIn{x0,x1,x2,x3} = 0 v 2 v 4]

which is much shorter and also easier to understand (it reads: "Either none, two or four attributes from among x0, x1, x2, x3 take value 1").

The above examples show that the counting attribute represents a very powerful descriptive concept. It allows one to express very concisely a wide range of relations for which an equivalent DNF expression would be very complex. If the counting attribute involves N binary attributes, it allows representation of all combinations of counts of N properties. There are 2^{N+1} such combinations (2^N subsets multiplied by two attribute values for each subset).

Figure 9 illustrates three out of the 32 possible concepts that can be expressed by one counting attribute #AttrIn{x0,x1,x2,x3}. In order to establish the value of a concept for a given instance (a cell in the diagram), the number of occurrences of attribute value "1" in the attribute vector representing this instance is counted. If the number of "1s" matches the concept definition then the instance belongs to the concept and is marked using "+"; otherwise it does not belong, and is marked by "-".



A. At least 3-of-4
[#AttrIn{x0,x1,x2,x3} >= 3]

B. 1 or 4-of-4
[#AttrIn{x0,x1,x2,x3} = 1,4]

C. Even-of 4
[#AttrIn{x0,x1,x2,x3} = 0,2,4]

Figure 9. Examples of M-of-N concepts that can be represented by using the counting attribute #AttrIn{x0,x1,x2,x3}.

The counting attribute generation rule represents a heuristic for changing the representation space. The following section justifies the rule by showing a multistage process of constructing counting attributes.

3.5 Relating Logic To Arithmetic

The underlying idea of the transition from logic-type to arithmetic-type descriptions goes back to the link between the binary system and digital computers. For several decades logic circuits not only facilitated logical operations but also arithmetic ones. Actually, all possible arithmetic operations, starting with addition through multiplication, ending with the most complex functions, are in fact represented by logic circuits. A link between logic and arithmetic could be expressed by two logical operators XOR and AND on one side and the arithmetic operator ADD on the other. The truth table in Fig. 10a shows the relationship between the two logical operators and the arithmetic operator. Addition of two binary digits can be expressed using Carry and Sum. Carry can be represented as logical AND, and Sum can be represented as logical XOR. Other arithmetic operators, such as multiplication, subtraction, can be expressed using the ADD operator.

The truth table from Fig. 10a is equivalent with the diagrams in Fig. 10b and 10c. The diagram in Fig. 10b shows how the ADD operator is represented using Carry and Sum. In a similar way, Fig. 10c shows how it is possible with AND and XOR operators. Finally, Fig. 10d is an abstraction of the XOR-pattern in the binary domain.

(a)

		ADD	
		Carry	Sum
x	y	AND	XOR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) **(c)** **(d)**

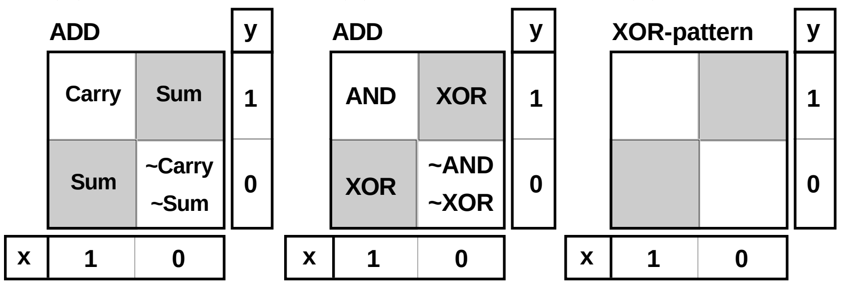
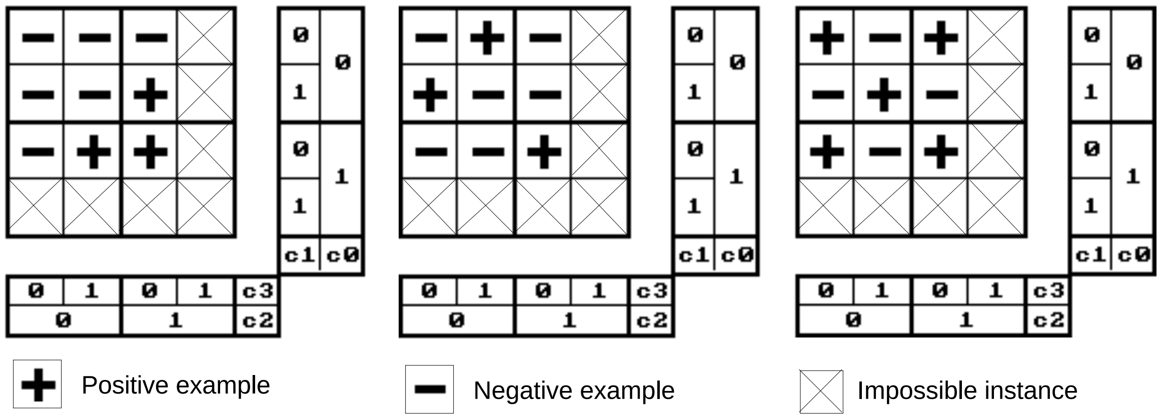


Figure 10. (a) Truth table for addition of two binary digits and its relation to the AND and XOR operators. (b) Visualization of the ADD operator using Carry and Sum. (c) Visualization of the ADD operator using AND and XOR. (d) Visualization of an XOR-pattern.

It is easy to see that an XOR concept, determined by the XOR operator, could be read as *Exactly 1-of-2* concept. The equivalence of the logic and arithmetic operators is useful in representing M-of-N concepts, which generalize the XOR concept. The logic representation of M-of-N concepts exhibits the XOR symmetry, and can be detected by searching for XOR-patterns in the data or hypotheses (Fig. 10d). XOR symmetry implies that instances described by a rule (x & ~y or y & ~x) belong to the same class. Using a new attribute, XOR-attribute, defined as (x & ~y or y & ~x) one can merge areas of the same class membership. However, without removing the original attributes, the representation space would just grow. In order to replace the original attributes with the one describing XOR relationship, one also needs to add an additional attribute describing either (x & y) or (~x & ~y) cell. Let us assume that (x & y) is used to define AND-attribute. The remaining fourth cell in the diagram can be uniquely represented as negation of the XOR and AND attributes (Fig 10c).

Fig. 11 shows such a transformation of the representation space and the three concepts defined in Fig. 9 (note the similar arrangement of the instances in both representation spaces). Attributes c0 and c2 represent AND-attributes, and c1 and c3 represent XOR-attributes. Because the same number of attributes is replacing the original ones the size of the representation space remains unchanged. However, since areas described by true values of XOR-attributes have merged, therefore the number of instances has decreased. The crossed cells represent impossible combinations of values of the new attributes. The diagrams identify those areas as (c0 & c1) or (c2 & c3), which means that instances described by true values of XOR-attribute and AND-attribute are impossible (the same way as Carry=1 and Sum=1 is irrelevant in defining ADD, Fig. 10a).

The impossible areas can be easily removed by combining AND and XOR attributes into ADD attributes (Fig. 12). In order to facilitate this operation, boolean values are replaced by integers. Hence, the new attributes are multivalued. In this representation space, it is easy to observe again the symmetry of the three concepts. This allows further transformations, i.e., merging symmetrical subconcepts and replacing AND and XOR with ADD. The final representation spaces with the three concepts is given in Fig. 13. Concept examples in the final representation space (Fig. 13) have direct relationship with examples in former representation spaces, e.g. in Fig. 12 (note matching patterns of related examples).



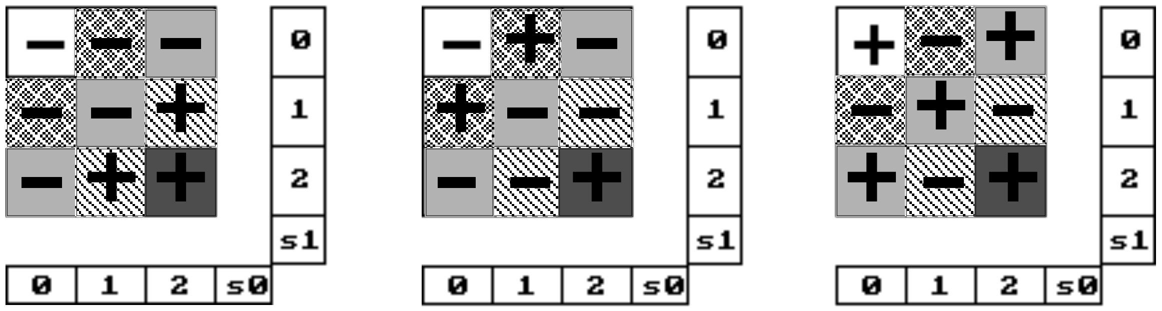
A. At least 3-of-4 **B.** 1 or 4-of-4 **C.** Even-of 4

The new attributes are defined as follows:

c0 <:: x0 & x1 **c1** <:: x0 & ~x1 or x1 & ~x0 **c3** <:: x2 & ~x3 or x3 & ~x2

c2 <:: x2 & x3

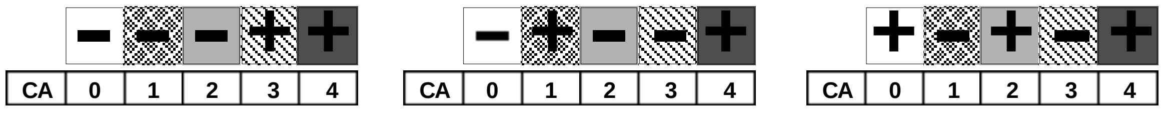
Figure 11. Transformation merging symmetrical subconcepts.



A. At least 3-of-4 **B.** 1 or 4-of-4 **C.** Even-of 4

The new attributes are defined as follows: **s0** <:: x0 + x1 **s1** <:: x2 + x3

Figure 12. Transformation replacing AND and XOR with ADD. The concepts are symmetrical with respect to attributes s0 and s1. The shaded areas mark XOR symmetries in this representation space.



A. At least 3-of-4 **B.** 1 or 4-of-4 **C.** Even-of 4

CA <:: s0 + s1 (or in terms of original attributes, **CA** <:: x0 + x1 + x2 + x3)

Figure 13. The three concepts in the final representation space.

In summary, the counting attribute generation rule allows direct transformation from the representation space with XOR-patterns (Fig. 9) to the representation space using counting attributes (Fig. 13).

3.6 Learning Conditional M-of-N Rules

The algorithm for building concept descriptions involving counting attributes is based on detecting XOR patterns in the DNF expression of the concept to be learned. The DNF description is determined using an AQ-based rule learning program. Attributes constituting XOR patterns are grouped into maximum symmetry classes. For each MSC-class the counting attribute generation rule is applied. The algorithm is presented in Fig. 14.

```
1.   Determine a DNF concept description from training examples projected into the current representation
     space.  If the expression is "sufficiently"  simple, STOP.
2.   Detect XOR-patterns in the learned concept description.
3.   If XOR-patterns do not exist, then STOP. Otherwise:
     Build maximum symmetry classes (MSCs). For each MSC-class, introduce a "counting attribute" and add
     the attribute to the representation space. Project the training data into the new representation space.
     Go to step 1.
```

Figure 14. Algorithm for changing the representation space based on XOR-patterns.

Suppose, for example, the following patterns were detected, x1 XOR x3, x1 XOR x5, and x1 XOR x7. Then the MSC-class is {x1, x3, x5, x7}. For each MSC-class a "counting attribute" is created. In the above example, such an attribute is #AttrIn{x1,x3,x5,x7}. ("The number of attributes from the set {x1,x3,x5,x7} that take value true"). Its domain is an integer interval from 0 to 4. Conditions involving counting attributes can represent arbitrary internal disjunction of values of the attribute. Thus, a counting attribute replaces a group of XOR relations connected by the transitivity relation.

If a learning system uses different representational formalisms for data and hypotheses (e.g. decision tree based systems) then detecting XOR-patterns in hypotheses may be difficult. In such situations, XOR patterns may be detected by the data-driven approach. In AQ learning systems that use VL1 description language both for data and rules, functional-patterns can be detected either by a data-driven or a hypothesis-driven approach. An examination of descriptions generated by FOIL (Quinlan, 1990) for MONK2 and similar problems indicates that they contain XOR-patterns. The proposed functional-patterns go beyond intra-construction and inter-construction operators used in Duce and CIGOL systems (Muggleton, 1987; Muggleton & Buntine, 1988), which are forms of rule-patterns in Horn-clauses (Wnek & Michalski, 1994b).

4 ILLUSTRATIVE EXAMPLE: MONK2 PROBLEM

The concept to be learned is the MONK2 problem (Thrun et al., 1991; Wnek & Michalski, 1994a). Figure 15A shows a diagram visualizing the problem. The total number of possible instances in the representation space is 432. In the diagram, the target concept is represented by 142 instances (shaded area). The remaining 290 instances represent the negation of the concept. The training set is represented by 64 positive (+) and 105 negative (-) examples. The data contains no noise.

4.1 Learning In The Original Representation Space

The MONK2 problem is hard for symbolic learning systems. In fact, none of the 18 symbolic learners taking part in the international competition learned the MONK2 concept (Thrun et al., 1991). This problem is also hard for the AQ15 program.

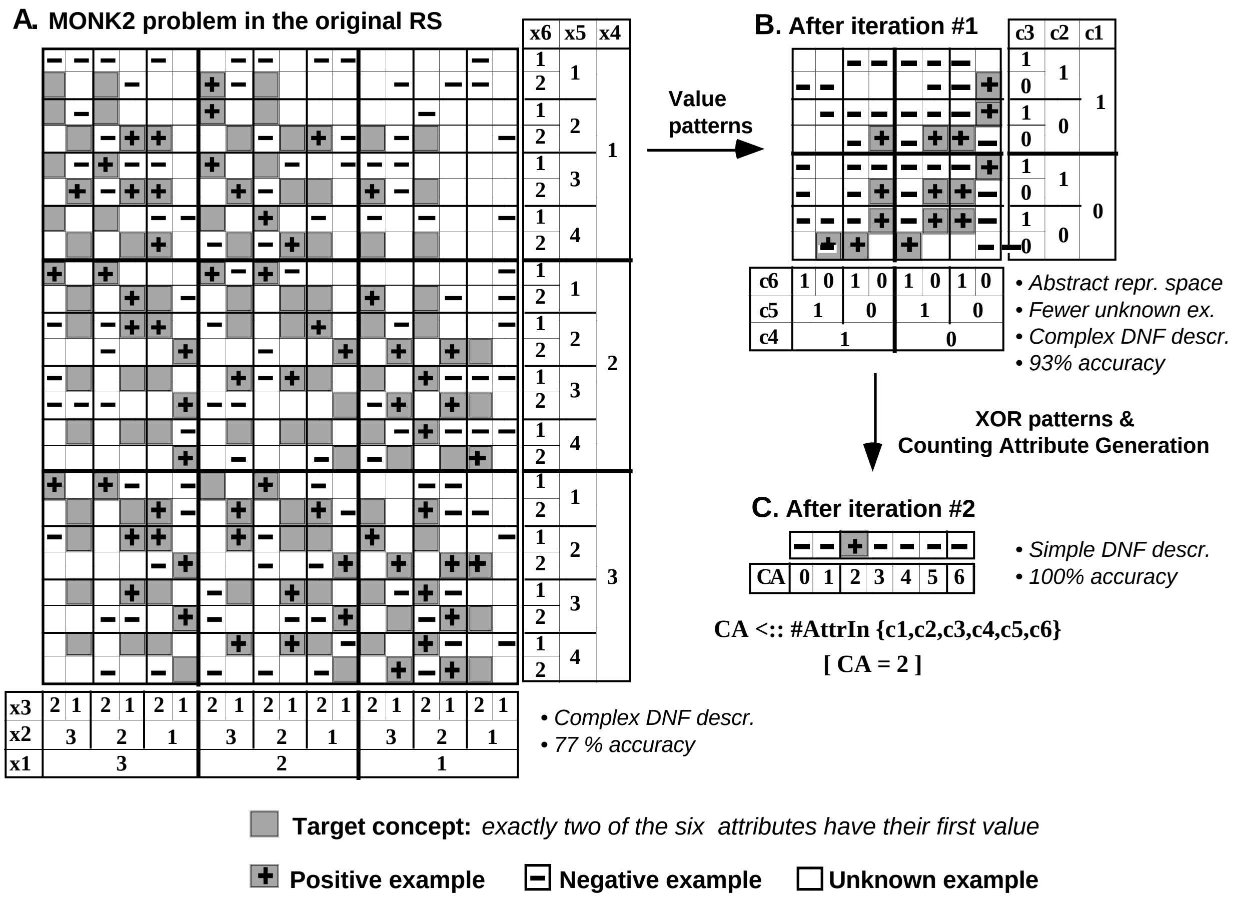


Figure 15. Learning the MONK2 problem in two iterations of the AQ-HCI method

The descriptions generated may slightly vary with different parameter settings but all have the following characteristics: they are inaccurate (about 75% prediction accuracy), they consist of many rules, and the rules use many conditions. The program's output[1] is presented in Fig. 16. As many as 16 rules generalize only 64 positive examples. Almost all rules involve all six attributes in describing the concept. It means that all attributes are equally important in the concept description, and moreover, the logical operators (and, or) are not capable of capturing meaningful relationships.

4.2 Representation Space Transformation: Iteration #1

Descriptions produced by AQ15 in the original representation space contain certain patterns that are easy to detect. Many conditions involving the same attribute tend to use the same set of attribute values. In other words they form value-patterns (Wnek and Michalski, 1994a). For example, conditions involving attribute HS use the following groupings of values {s,o} nine times, {r} four times, {r,o} and {o} once. Taking into consideration that HS attribute has three values {r,s,o}, this value-pattern suggests that the division of this set into two subsets, {r} and {s,o} should be meaningful. Similarly, values of other attributes can also be grouped. The AQ-HCI method

[1] The following parameters were used: disjoint covers mode, so the generated positive and negative descriptions are disjoint over areas not represented by training examples; the most specific rules are generated; during rule generation 10 alternative solutions are considered (beam width); among those rules, one rule is selected that best satisfies the default criteria, i.e. maximizes the number of newly covered (not covered by previous rules) examples.

changes the representation space according to the following new attribute definitions (Fig. 17). (To contrast the new representation with the original one we also substitute new attributes for SM and TI. Since these attributes are binary this is only a change in names.)

The transformed learning task is visualized in Fig. 15B. The new representation space has become significantly smaller, there are only 64 instances in the new space vs. 432 instances in the original representation space. The number of attributes is the same but all of them are binary.

1	[HS=r]	&	[BS=s,o]	&	[SM=y]	&	[HO=f,b]	&	[JC=y,g,b]	&	[TI=n]		(t:9, u:9)
2	[HS=s,o]	&	[BS=s,o]	&	[SM=y]	&	[HO=f,b]	&	[JC=y,g,b]	&	[TI=y]		(t:9, u:9)
3	[HS=s,o]	&	[BS=s,o]	&	[SM=n]	&	[HO=f,b]	&	[JC=r]	&	[TI=y]		(t:7, u:7)
4	[HS=r]	&	[BS=s,o]	&	[SM=n]	&	[HO=f,b]	&	[JC=y,g,b]	&	[TI=y]		(t:5, u:5)
5	[HS=r,o]	&	[BS=r]	&			[HO=s,f]	&	[JC=g,b]	&	[TI=n]		(t:5, u:4)
6	[HS=s,o]	&	[BS=r,o]	&	[SM=y]	&	[HO=s,b]	&	[JC=g]	&	[TI=n]		(t:4, u:4)
7	[HS=s,o]	&	[BS=s,o]	&	[SM=n]	&	[HO=s]	&	[JC=y,g,b]	&	[TI=y]		(t:4, u:4)
8	[HS=s,o]	&	[BS=r]	&	[SM=y]	&	[HO=f,b]	&	[JC=y]	&	[TI=n]		(t:4, u:4)
9			[BS=r,o]	&	[SM=n]	&	[HO=s]	&	[JC=y,g]	&	[TI=n]		(t:4, u:3)
10	[HS=s,o]	&	[BS=r,o]	&	[SM=n]	&	[HO=s,b]	&	[JC=r]	&	[TI=n]		(t:3, u:3)
11	[HS=s,o]	&	[BS=r]	&	[SM=n]	&	[HO=f,b]	&	[JC=y]	&	[TI=y]		(t:3, u:3)
12	[HS=s,o]	&	[BS=s,o]	&	[SM=y]	&	[HO=f,b]	&	[JC=r]	&	[TI=n]		(t:2, u:2)
13	[HS=r]	&	[BS=s,o]	&	[SM=n]	&	[HO=f,b]	&	[JC=r]	&	[TI=n]		(t:2, u:2)
14	[HS=s,o]	&	[BS=s]	&	[SM=y]	&	[HO=s]	&	[JC=y,b]	&	[TI=n]		(t:2, u:2)
15	[HS=o]	&	[BS=s]	&	[SM=y]	&	[HO=s]	&	[JC=g]	&	[TI=n]		(t:1, u:1)
16	[HS=r]	&	[BS=r]	&	[SM=n]	&	[HO=b]	&	[JC=y]	&	[TI=n]		(t:1, u:1)

Figure 16. The MONK2 Concept Learned in the Original Representation Space

(c1 = 1) <:: [HS=r]	(c2=1) <:: [BS=r]	(c3=1) <:: [SM=y]
(c1 = 0) <:: [HS=s,o]	(c2=0) <:: [BS=s,o]	(c3=0) <:: [SM=n]
(c4 = 1) <:: [HO=s]	(c5=1) <:: [JC=r]	(c6=1) <:: [TI=y]
(c4 = 0) <:: [HO=f,b]	(c5=0) <:: [JC=y,g,b]	(c6=0) <:: [TI=n]

Figure 17. Attributes Constructed From Value-patterns

The number of instances representing the target concept is 15, therefore in the worst case, the number of rules required to describe the concept is 15. This is a reduction in description complexity in comparison to the original representation space. Each instance in the new space represents from 1 to 24 instances that were mapped from the original space. The transformation does not cause ambiguity in the new representation space, i.e., each new instance represents instances of the same class, either positive or negative. For more details see (Wnek, 1993).

In this representation space, all possible positive examples are present, and only 13 negative examples are missing (in the original space only 64 positive examples out of 142 are present). It seems that learning should give better results. However, the AQ15 learning program still generates a long and inaccurate description of the concept (Fig. 18). Errors are caused by the overly general rule #1. This rule covers not only two positive examples but also covers two negative instances.

4.3 Representation Space Transformation: Iteration #2

The description obtained after the first transformation of the representation space is more accurate but still very complex (Figs. 18, 15B). Therefore, the search for a better representation is continued, and the XOR-patterns are found. Fig. 19 lists examples of pairs of rules with XOR-patterns present. All six attributes form a MSC class. From this class a new counting attribute is constructed. It is defined as #AttrIn{c1,c2,c3,c4,c5,c6}. Its domain is an integer interval from 0 to 6. Summing up values in the XOR-patterns always gives the exact value 2. The final concept description is [#AttrIn{c1,c2,c3,c4,c5,c6} = 2], i.e., exactly two of the six attributes are present. Fig. 15C visualizes the final representation space and the concept learned.

1	[c1=0]	&			[c3=0]	&			[c5=1]	&	[c6=0]	(t:2, u:2)
2	[c1=1]	&	[c2=1]	&	[c3=0]	&	[c4=0]	&	[c5=0]	&	[c6=0]	(t:1, u:1)
3	[c1=1]	&	[c2=0]	&	[c3=1]	&	[c4=0]	&	[c5=0]	&	[c6=0]	(t:1, u:1)
4	[c1=1]	&	[c2=0]	&	[c3=0]	&	[c4=1]	&	[c5=0]	&	[c6=0]	(t:1, u:1)
5	[c1=1]	&	[c2=0]	&	[c3=0]	&	[c4=0]	&	[c5=0]	&	[c6=1]	(t:1, u:1)
6	[c1=0]	&	[c2=1]	&	[c3=1]	&	[c4=0]	&	[c5=0]	&	[c6=0]	(t:1, u:1)
7	[c1=0]	&	[c2=1]	&	[c3=0]	&	[c4=1]	&	[c5=0]	&	[c6=0]	(t:1, u:1)
8	[c1=0]	&	[c2=1]	&	[c3=0]	&	[c4=0]	&	[c5=0]	&	[c6=1]	(t:1, u:1)
9	[c1=0]	&	[c2=0]	&	[c3=1]	&	[c4=1]	&	[c5=0]	&	[c6=0]	(t:1, u:1)
10	[c1=0]	&	[c2=0]	&	[c3=1]	&	[c4=0]	&	[c5=1]	&	[c6=0]	(t:1, u:1)
11	[c1=0]	&	[c2=0]	&	[c3=1]	&	[c4=0]	&	[c5=0]	&	[c6=1]	(t:1, u:1)
12	[c1=0]	&	[c2=0]	&	[c3=0]	&	[c4=1]	&	[c5=0]	&	[c6=1]	(t:1, u:1)
13	[c1=0]	&	[c2=0]	&	[c3=0]	&	[c4=0]	&	[c5=1]	&	[c6=1]	(t:1, u:1)
14	[c1=1]	&	[c2=0]	&	[c3=0]	&	[c4=0]	&	[c5=1]	&	[c6=0]	(t:1, u:1)

Figure 18. The Concept Learned in the Representation Space Developed in Iteration #1

Rule No						
3	**[c1=1]**	**[c2=0]**	[c3=1]	[c4=0]	[c5=0]	[c6=0]
6	**[c1=0]**	**[c2=1]**	[c3=1]	[c4=0]	[c5=0]	[c6=0]
3	[c1=1]	[c2=0]	**[c3=1]**	[c4=0]	[c5=0]	**[c6=0]**
5	[c1=1]	[c2=0]	**[c3=0]**	[c4=0]	[c5=0]	**[c6=1]**
2	**[c1=1]**	[c2=1]	[c3=0]	**[c4=0]**	[c5=0]	[c6=0]
7	**[c1=0]**	[c2=1]	[c3=0]	**[c4=1]**	[c5=0]	[c6=0]
6	[c1=0]	**[c2=1]**	[c3=1]	[c4=0]	**[c5=0]**	[c6=0]
10	[c1=0]	**[c2=0]**	[c3=1]	[c4=0]	**[c5=1]**	[c6=0]
10	**[c1=0]**	[c2=0]	**[c3=1]**	[c4=0]	[c5=1]	[c6=0]
14	**[c1=1]**	[c2=0]	**[c3=0]**	[c4=0]	[c5=1]	[c6=0]

Figure 19. An Example of XOR-patterns Leading to the Creation of the Attribute #AttrIn{c1,c2,c3,c4,c5,c6}

5 EXPERIMENTS ON LEARNING CONDITIONAL M-of-N RULES

To test the presented ideas, the AQ-HCI system was applied to learning several concepts that can be simply represented as conditional M-of-N rules. The first experiment involved the MONK2 problem (Thrun et al., 1991). The MONK2 problem, which presented such a significant difficulty for many machine learning programs (Thrun et al., 1991), was solved with 100% accuracy, and the minimal complexity of the description.

Concept:	**MONK2: "Exactly two of the six attributes have their first value"** [#AttrIn{First(a1), First(a2), ..., First(a6)} = 2]	
Domain:	6 multivalued attributes (a1, ..., a6)	
Training set	142 examples out of 432	
AQ15:	Prediction accuracy: 77%	Complexity: 16 rules
AQ17-HCI:	Prediction accuracy: 100%	Complexity: 1 rule

Figure 20. Results from learning the MONK2 concept.

Further experiments concerned learning concepts involving 6, 9, 10, and 16 binary attributes. Figure 21 summarizes results from the experiments.

Let	x0, x1, x2, x3, x4, x5, x6, x7, x8, x9, x10, x11, x12, x13, x14, x15, be binary attributes S1 = {x0,x2,x4,x6,x8}, S2 = {x1,x3,x5,x7}, S3 = {x12,x13,x14,x15}, be sets of attributes	
Concept:	**"At least 3 in S1 and at least 2 in S2"** [#AttrIn{x0,x2,x4,x6,x8} > = 3] & [#AttrIn{x1,x3,x5,x7} >= 2]	
Domain:	9 binary attributes (x0, x1, ..., x8)	
Training set	50% of all examples	
AQ15:	Prediction accuracy: 78%	Complexity: 42 rules
AQ17-HCI:	Prediction accuracy: 100%	Complexity: 1 rule
Concept:	**"At least 3 in S1, at least 2 in S2, and x9 holds"** [#AttrIn{x0,x2,x4,x6,x8}>=3] & [#AttrIn{x1,x3,x5,x7} >= 2] & [x9= 1]	
Domain:	10 binary attributes (x0, x1, ..., x9)	
Training set	20% of all examples	
AQ15:	Prediction accuracy: 96 %	Complexity: 22 rules
AQ17-HCI:	Prediction accuracy: 100%	Complexity: 1 rule
Concept:	**"At least 3 in S1, at least 2 in S2, and x9 holds or exactly 2 in S3"** [#AttrIn{x0,x2,x4,x6,x8} >= 3] & [#AttrIn{x1,x3,x5,x7}>= 2] & [x9= 1] or [#AttrIn{x12,x13,x14,x15} = 2]	
Domain:	16 binary attributes (x0, x1, ..., x15)	
Training set	10% of all examples	
AQ15:	Prediction accuracy: 93%	Complexity: 92 rules
AQ17-HCI:	Prediction accuracy: 100%	Complexity: 5 rules

Figure 21. Summary of experiments on learning conditional M-of-N rules.

The experiments show that AQ-HCI was very effective in learning these concepts. Testing on more complex conditional M-of-N concepts indicates radical improvement in both prediction accuracy and simplicity of the descriptions, as compared to the AQ15 system.

6 CONCLUSION

The presented method can be applied to a wide range of domains where logical conditions need to be combined with simple arithmetic relations to capture the essence of the target concept. Such domains include economics, medicine, computer vision, biochemistry.

Concepts of this type are very difficult to learn by conventional learning methods. The proposed method learns such concepts by searching for XOR symmetry patterns in initially created DNF descriptions, and this is a form of a hypothesis-driven constructive induction. Detected patterns are used to create "counting attributes" that enhance the knowledge representation space. Extending the representation space by such dimensions is a form of a task-oriented adaptation of the space. Initial experiments have shown that the method is very effective in learning conditional M-of-N rules.

Future research needs to investigate the scope of problems for which this method is most effective and the convergence of the proposed algorithm.

REFERENCES

Baffes, P.T. & Mooney, R.J. (1993). Symbolic Revision of Theories with M-of-N Rules. In *Proceedings of the 2nd International Workshop on Multistrategy Learning* (pp. 69-75). Harpers Ferry, WV: Morgan Kaufmann.

Bloedorn, E. & Michalski, R.S. (1991). Data Driven Constructive Induction in AQ17-PRE: A Method and Experiments. In *Proceedings of the Third International Conference on Tools for AI* (pp. 25-35). San Jose, CA.

Callan, J.P. & Utgoff, P.E. (1991). A Transformational Approach to Constructive Induction. In *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 122-126). Evanston, IL: Morgan Kaufmann.

Fawcett, T.E. & Utgoff, P.E. (1991). A Hybrid Method for Feature Generation. In *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 137-141). Evanston, IL: Morgan Kaufmann.

Jensen, G.M. (1975). *Determination of Symmetric VL1 Formulas: Algorithm and Program SYM4*. Master's thesis. (Tech. Rep. No. UIUCDCS-R-75-774). Urbana-Champaign: University of Illinois, Department of Computer Science.

Michalski, R.S. (1969). Recognition of Total or Partial Symmetry in a Completely or Incompletely Specified Switching Function. In *Proceedings of the IV Congress of the International Federation on Automatic Control (IFAC)*, *27*, 109-129.

Michalski, R.S. (1975). "Variable-valued Logic and Its Application to Pattern Recognition and Machine Learning." In D. Rine (Ed.) *Computer Science and Multiple-Valued Logic Theory and Applications*, North-Holland Publishing.

Michalski, R.S. (1983). A Theory and Methodology of Inductive Learning. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Palo Alto CA: TIOGA Publishing.

Michalski, R.S., Mozetic, I., Hong, J. & Lavrac, N. (1986). The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains. In *Proceedings of AAAI-86* (pp. 1041-1045). San Mateo, CA: Morgan Kaufmann.

Michalski, R.S., Rosenfeld, A. & Aloimonos, Y. (1994). *Machine Vision and Learning: Research Issues and Directions* (Tech. Rep. No. MLI 94-6). Fairfax, VA: George Mason University, Center for Machine Learning and Inference. (Tech. Rep. No. CAR-TR-739, CS-TR-3358). College Park, MD: University of Maryland, Center for Automation Research.

Muggleton, S. (1987). Duce, an Oracle-Based Approach to Constructive Induction. In *Proceedings of IJCAI-87* (pp. 287-292). San Mateo, CA: Morgan Kaufmann.

Muggleton, S. & Buntine, W. (1988). Machine Invention of First Order Predicates by Inverting Resolution. In *Proceedings of the 5th International Conference on Machine Learning* (pp. 339-352). San Mateo, CA: Morgan Kaufmann.

Murphy, P. M. & Pazzani, M. J. (1991). ID2-of-3: Constructive Induction of M-of-N Concepts for Discriminators in Decision Trees. In *Proceedings of the 8th International Workshop on Machine Learning* (pp. 183-187). Evanston, IL: Morgan Kaufmann.

Quinlan, J.R. (1990). Learning Logical Definitions from Relations. *Machine Learning, 5*, 239-266.

Seshu, R. (1989). Solving the Parity Problem. In *Proceedings of EWSL-89* (pp. 263-271). Montpellier, France.

Spackman, K.A. (1988). Learning Categorical Decision Criteria in Biomedical Domains. In *Proceedings. of the 5th International Conference on Machine Learning* (pp. 36-46). San Mateo, CA: Morgan Kaufmann.

Thrun, S.B., Bala, J., Bloedorn, E., Bratko, I., Cestnink, B., Cheng, J., DeJong, K.A., Dzeroski, S., Fahlman, S.E., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R.S., Mitchell, T., Pachowicz, P., Vafaie, H., Van de Velde, W., Wenzel, W., Wnek, J. & Zhang, J. (1991). *The MONK's Problems: A Performance Comparison of Different Learning Algorithms*. (Tech. Rep. October). Pittsburgh, PA: Carnegie Mellon University, School of Computer Science.

Towell, G. G. & Shavlik, J. W. (1994). Refining Symbolic Knowledge Using Neural Networks. In R.S. Michalski & G. Tecuci (Eds.), *Machine Learning: A Multistrategy Approach, Vol. IV* (pp. 405-429). San Mateo, CA: Morgan Kaufmann.

Wnek, J. (1993). Hypothesis-driven Constructive Induction. (Doctoral Dissertation, George Mason University, School of Information Technology and Engineering, Fairfax, VA). Ann Arbor, MI: University Microfilms Int.

Wnek, J. & Michalski, R.S. (1994a). Comparing Symbolic and Subsymbolic Learning: Three Studies. In R.S. Michalski & G. Tecuci (Eds.), *Machine Learning: A Multistrategy Approach, Vol. 4*, San Mateo, CA: Morgan Kaufmann.

Wnek, J. & Michalski, R.S. (1994b). Hypothesis-driven Constructive Induction in AQ17-HCI: A Method and Experiments. *Machine Learning, 14*, 139-168.

Wnek, J. & Michalski, R.S. (1994c). Discovering Representation Space Transformations for Learning Concept Descriptions Combining DNF and M-of-N Rules. In *Working Notes of the ML-COLT'94 Workshop on Constructive Induction and Change of Representation* (pp. 61-68). New Brunswick, NJ.