Semantic and Syntactic Attribute Types in AQ Learning

Ryszard S. Michalski and Janusz Wojtusiak

MLI 07-1
P 07-1
March 29, 2007

SEMANTIC AND SYNTACTIC ATTRIBUTE TYPES IN AQ LEARNING

Ryszard S. Michalski* and Janusz Wojtusiak
Machine Learning and Inference Laboratory
George Mason University
Fairfax, VA 22030-4444, USA

*Also with the Institute of Computer Science,
Polish Academy of Sciences

{michalski, jwojt}@mli.gmu.edu
http://www.mli.gmu.edu

Abstract

AQ learning strives to perform *natural induction* that aims at deriving general descriptions from specific data and formulating them in human-oriented forms. Such descriptions are in the forms closely corresponding to simple natural language statements, or are transformed to such statements in order to make computer generated knowledge easy to interpret and understand. An important feature of natural induction is that it employs a wide range of types of attributes to guide the process of generalization. Attribute types constitute problem domain knowledge, and are provided by the user, or are inferred by the learning program from the data. This paper makes a distinction between semantic and syntactic attribute types in AQ learning, explains their relationships and provides their classifications. Semantic types depend solely on the structure of attribute domains and help to create plausible generalizations, while syntactic types depend also on physical properties of attribute domains, and are used to efficiently implement semantic types.

Keywords: Attribute types, Machine learning, AQ learning, Natural induction, Computational Learning.

Acknowledgments

The authors thank Ken Kaufman and Jarek Pietrzykowski for proofreading and for their comments that helped to improve this paper.

This research has been conducted in the Machine Learning and Inference Laboratory of George Mason University, whose research activities have been supported in part by the National Science Foundation Grants No. IIS 9906858 and IIS 0097476, and in part by the UMBC/LUCITE #32 grant. The findings and opinions expressed here are those of the authors, and do not necessarily reflect those of the above sponsoring organization.

1 INTRODUCTION

The AQ learning methodology pioneered the *progressive covering*, also known as *separate and conquer* approach to inductive rule learning. The most distinctive feature of AQ learning in comparison to other learning methods is that it strives to implement *natural induction*, by which we mean an inductive learning process that aims at creating general concept descriptions from concept examples or discovering patterns in data, and expressing them in human-oriented forms (Michalski, 1999). Such descriptions are in the forms resembling simple natural language statements, or are transformed to such statements in order to make computer generated knowledge easy to interpret and understand. Unlike typical machine learning programs that emphasize mainly predictive accuracy of the learned knowledge, natural induction gives equal importance to predictive accuracy and to the interpretability of that knowledge.

Natural induction is implemented in AQ learning by employing *attributional calculus* as a representation language and an underlying logic system (Michalski, 2004). To facilitate natural induction, attributional calculus combines elements of propositional, predicate, and multiple-valued logic, and includes additional forms that resemble those used in natural language descriptions. Results form learning are in the form of *attributional classifiers,* in which each concept (or decision class) is represented by an *attributional ruleset*. Such a ruleset is equivalent to a DNF (disjunctive normal form) whose components (conditions, a.k.a. selectors) directly correspond to simple natural language statements.

Matching attributional classifiers against concept instances can be done *crisply*, by treating individual rules as binary logic expressions that produce a "match" or "no-match", or *flexibly*, by considering them as continuously-valued expressions that produce a degree of match (that varies between 0 and 1).

Attributional classifiers are more expressive than conventional decision rules (in which conditions are in the form <attribute-relation-value>), *m*-of-*n* rules, decision trees, and k-nearest neighbors. They are more general because they employ a larger set of operators, and use different methods of matching expressions with instances than conventional representations. By employing a more expressive representation, an AQ learning program can hypothesize simpler concept descriptions (e.g., consisting of a smaller number of rules) than other rule learning programs. This ability is achieved, however, at the expense of higher complexity of the process (searching through significantly larger spaces of possible descriptions). In order to streamline this search, AQ learning distinguishes between many different types of attributes, and uses the type information in the process of generalization.

This paper makes a distinction between semantic and syntactic attribute types, provides their classifications, and describes data structures for their efficient implementation in the AQ21 learning program (Wojtusiak et al., 2006). We start by explaining semantic types.

2 SEMANTIC ATTRIBUTE TYPES

Semantic attribute types depend solely on the structure of the attribute domain, and have been defined in attributional calculus ($\mathcal{AC}$). To make this paper self-explanatory, we briefly review these types, borrowing their description from (Michalski, 2004).

Most of the attribute types defined in $\mathcal{AC}$ correspond to standard measurement scales, such as nominal, ordinal, interval, ratio, and absolute. In addition to these types, $\mathcal{AC}$ includes also *cyclic*, *structured, set-valued,* and *compound* types (Table 1). During learning, attribute types are used to guide the process of plausible generalization, according to the *rules of inductive generalization* introduced in (Michalski, 1983). Attribute types constitute a form of domain knowledge, and are specified by the user defining the learning problem to the program.

Table 1: A Brief Explanation of Semantic Attribute Types Defined in $\mathcal{AC}$
and Recognized in AQ Learning (based on Michalski, 2004)

- ♦ *nominal* (a.k.a. *categorical*), if the attribute domain is an unordered set (for example, "Blood type," "Person's name").
- ♦ *linear*, if its domain is a totally ordered set. Linear attributes are in turn classified into *ordinal*, *interval*, *ratio*, and *absolute*, corresponding to the common measurement scales—see Table 2 (for example, a student's grade is an ordinal attribute, an object's temperature in degrees Celsius is an interval attribute, an object's weight is a ratio attribute, and the number of atoms in a molecule is an absolute-type attribute).
- ♦ *cyclic*, if its domain is a cyclically ordered set (for example, Hours of the day, Days of the week, Months of the year, Signs of the zodiac, Time zones, etc.).
- ♦ *structured* (a.k.a. *hierarchical*), if its domain is a hierarchically ordered set (for example, a type- or generalization-hierarchy of geometrical shapes, plants, animals, diseases, military ranks, airplanes, etc.).
- ♦ *set-valued,* if its domain is the power set of a *base set*, and its values are subsets of the base set. Set-valued attributes can be represented by a set of binary attributes. It is advantageous to use set-valued attributes when the base set is very large, but the subsets constituting their frequent values are small. For example, a value may be the set of diseases or manifestations recognized in a patient, and the base will be the set of known diseases, or the value may be a set of items one bought is a department store (an itemset in data mining).
- ♦ *compound*, if its domain is the Cartesian product of the domains of attributes that are non-compound. A compound attribute is used to characterize an object or parts of an object in terms of *constituent* attributes that apply only to this object or to individual parts. Such a characterization is done by listing values of constituent attributes for this object or for a specific part, but without stating attribute names. The name of the object or the part is used as the name of the compound attribute. For example, "Weather" can be used as a compound attribute whose values are lists of properties typically used to characterize weather. For example, attributional calculus allows one to create an expression "Weather = sunny & humid", where "Weather" is the compound attribute and "sunny" and "humid" are values of its constituent attributes. As one can see, this expression directly corresponds to an equivalent natural language statement "The weather is sunny and humid", and appears to be more natural to people than a standard logic expression such as "The weather-type is sunny and the humidity is yes." Compound attributes thus facilitate natural induction.

The classification of attributes into nominal, ordinal, interval and ratio was introduced in a classic paper by Stevens (1946). The "structured" and "cyclic" types were introduced in (Michalski, 1978) to facilitate processes of generalizing or specializing descriptions using attributes of these types. The compound attribute was introduced in (Michalski, 2004).

The interval, ratio and absolute attributes are called *quantitative* (or *metric* or *numeric*), and the nominal, ordinal, cyclic, and structured attributes are called *qualitative* (or *non-metric* or *symbolic*). Although there has been some criticism of Stevens' classification of measurement scales from the statistical viewpoint (e.g. Velleman and Wilkinson, 1993), this criticism does not apply to AQ learning, because it does not treat all the attributes as numerical, but reasons with them according to their type. Moreover, Stevens' scales, which are widely used in psychology, reflect the way people think and reason with different attributes, and therefore are suitable for natural induction.

Table 2 provides a characterization of the domain for each attribute type, and corresponding *invariant transformation* by which is meant a transformation of the attribute domains that does not cause any loss of information conveyed by values of the attribute.

Table 2: Domain Structure and Invariant Transformation for Each Attribute Type (reproduced from Michalski, 2004).

TYPE/SCALE	***DOMAIN***	***INVARIANT TRANSFORMATION***
Nominal/categorical	unordered set	isomorphic
Structured (or hierarchical)	partially ordered set	node-type dependent
Set-valued	partially ordered set	isomorphic
Ordinal	totally ordered set	monotonic
Cyclic	cyclically ordered set	monotonic-cyclical
Interval	totally ordered set	linear: y'= a + by
Ratio	totally ordered set	ratio: y'= ay
Absolute	totally ordered set	none
Compound	*their domains and invariant transformations depend on constituent attributes.*	

Let us use simple examples to explain the concept of invariant transformation. Suppose that the domain of a nominal attribute "color" is {red, blue, yellow}. By isomorphically mapping this domain to a three-element set of names of these colors in another language, for example, in Polish, which is {czerwony, niebieski, zolty}, we can use the attribute color with the new values without any loss of information, as long as the mapping is applied consistently. Meaningful statements involving nominal variables can use operators "equal" and "not-equal", e.g., color = red, or color ≠ blue. Using the internal disjunction operator defined in $\mathcal{AC}$, we can also write color = red v blue, to mean that the color is red or blue.

All the standard logical operators apply to such statements, such as negation, logical disjunction, logical conjunction, exclusive-or, and equivalence. Although one can map such a domain to a set of numbers, e.g., {1, 2, 3}, these numbers serve only as labels, and cannot be meaningfully used in arithmetical operations. For example, adding or multiplying numbers representing values of the attribute color, as, for example, in a paint-by-numbers set, are not meaningful operations.

In structured (or hierarchical) attributes, the partial order is defined by a generalization hierarchy (a.k.a. *type* or *is-a* hierarchy). In this hierarchy, a parent node represents a more general concept than its offspring nodes. Examples of such hierarchies include geometric shapes, geographic names, plants, animals, etc. Figure 1 presents a possible domain of the structured attribute "Shape." In addition to operators applicable to nominal attributes that check for attribute-value equality (attribute = value) or inequality (attribute ≠ value), other operators applicable to structured attributes include *structure-based generalization*, *specialization*, and *modification*.

The generalization operator, *climb-tree(A, V, k)*, replaces value *V* in a description involving attribute *A* by a more general value located *k* levels above it in the generalization hierarchy for this attribute. For example, *climb-tree(shape, obtuse, 1*) replaces value *obtuse triangle* by *triangle* in a statement involving the attribute *shape.*

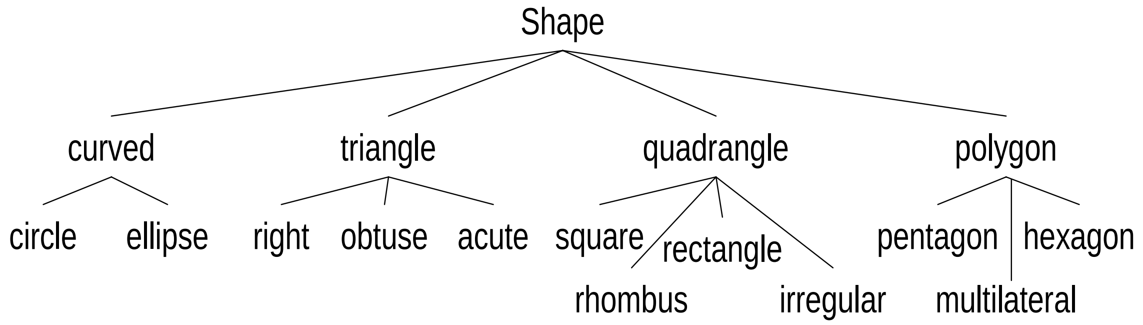

Figure 1: The domain of the structured attribute "shape" (reproduced from Michalski, 2004).

The operator *descend(A, V, k)* is an inductive specialization operator that replaces value *V* by one of the values located *k* levels down from *V*. The value is selected among alternatives either randomly or algorithmically. For example, the operator *descend(Shape, triangle, 1)* may randomly generate the value *right triangle,* as an instance of a triangle. An algorithmic application would direct the operator to choose, e.g., the first value on the list of descendents of the node "triangle", or a value that has some desirable property, e.g., is most frequently present in positive examples of a given concept.

Finally, the operator *select-sibling(A, V)* is a generate-alternative operator that replaces value *V* of attribute A by one of its sibling values in the hierarchy (again, the value can be selected randomly or algorithmically). For example, *select-sibling(Shape, square)* may produce "rectangle." As the above examples show, structured attributes facilitate operations of description generalization, specialization, and modification.

The domain of ordinal (a.k.a. rank) attributes is a totally ordered discrete set. For example, a course grade with possible values A, B, C, D or F is an ordinal attribute. A discretization of a continuous attribute creates an ordinal attribute. Because a discretized attribute is more abstract that the original one, a discretization is an abstraction operation. For example, suppose that the range of a continuous attribute "systolic blood pressure" has been discretized into {1, 2, 3}, where "1" stands for "low", "2" stands for "normal" and "3" stands for "high". If we perform a monotonic transformation of this domain into <L, N, H>, where "L" corresponds to "low", "N" to “normal”, and “H” to ‘high”, we can use the new domain without any loss of information, because values of ordinal attributes only convey information about the order of values, not about their magnitude.

Operators applicable to ordinal attributes include not only those applicable to nominal and structured attributes, but also relational operators, $<$, $\leq$, $>$, $\geq$, and the *range* operator, denoted by “..”, which defines a range of values. For example, if the domain of an ordinal attribute “strength” is {very_weak, weak, medium, strong, very_strong, mighty_strong}, we can write, *strength > medium*, to say that the value of *strength* is greater than *medium*. We can also write *strength = weak..very_strong* to say that the value is between *weak* and *very strong*, inclusive.

Set-valued attributes are useful for describing situations in which values of interest are small subsets of a large domain, called the *base set*. An example of a set-valued attribute is *itemset,* whose values are sets of goods customers purchase in a store. Individual customers usually purchase only a very small subset of goods available in a store (the latter is the base set). The structure of domains of set-valued attributes is a lattice. A set-valued attribute can be represented by a set of binary variables whose cardinality equals the cardinality of the base set. A value of such an attribute would be a characteristic vector identifying the subset of interest in the base set, in which “1” could indicate the presence, and “0” the absence of a value from the base set. Such a representation is not practical if the base set is very large. A better representation in such a case is to directly list values of the set-valued attribute. Values of set-valued attributes can be used as operands in set-theoretic operations, such as union, intersection, and set-difference, and in set-theoretic relations $A \subseteq B$, $A \subset B$, $A \not\subset B$, $A \cap B = \emptyset$. For example, given statements *itemset=A1* and *itemset=A2*, one can make statements such as $itemset = A1 \cup A2$, $itemset = A1 \cap A2$, or $itemset = A1 - A2$.

Cyclic attributes are like ordinal variables, but with the added assumption that the last value is followed by the first value in their domain. The operators $>$, $\geq$, $<$ and $\leq$ do not apply to these attributes, but the range operator “..” does. Cyclic attributes are used to describe cyclically occurring phenomena, e.g., days of the week, or months of the year. Thus, one can write: *day_of_week = Wed..Mon.*

Interval attributes are used to characterize quantities measured on a scale in which “0” is arbitrary, and values express counts of some arbitrary units from that zero point. For example, the temperature can measured on the Celsius scale (^oC), where “0” denotes the temperature of freezing water, and “100” denotes the temperature of boiling water, or the Fahrenheit scale (^oF), which is related to the Celsius scale by the linear function: $t^oC = 5/9\ (t^oF - 32)$, where t^oC and t^oF are temperatures measured on the Celsius scale and the Fahrenheit scale, respectively. Any other linear transformation of the domain will not change the meaning of the temperature measurement, as long as it is applied consistently, and units are specified.

All previously mentioned operators that apply to ordinal attributes apply also to interval attributes, as well as arithmetic operators “+” and “-“. For example, one can say that the difference between the temperature today, t_1, and yesterday, t_2, is $\Delta t\ ^oC = t_1\ ^oC - t_2\ ^oC$, which can be equivalently expressed as $F(\Delta t\ ^oC) = F(t_1\ ^oC) - F(t_2\ ^oC)$, where $F(t^oC)$ denotes the Fahrenheit temperature equivalent to the temperature in ^oC. Multiplication and division are *not* invariant operators for interval attributes. For example, $F(3 * t^oC) \neq 3 * F(t^oC))$. Thus, temperatures measured in ^oC or ^oF can be meaningfully added or subtracted, but not divided or multiplied. For example, if the temperature in a room rose from 35^{o}F to 70^{o}F, it does not mean the room is twice as hot.

Values of ratio attributes represent counts of some predefined units, as in interval attributes, but zero is not arbitrary; rather, it denotes the absolute minimum value of these attributes. Units can be, for example, kilograms or pounds for a weight attribute, inches or centimeters for a length attribute, etc. Different units are related by a multiplication coefficient. In addition to all the previous operators, ratio attributes can be also used in all arithmetic and algebraic operations.

Absolute variables represent direct counts of some items in a set (e.g., the number of people in a room). There have no units; and no transformations are allowed. Absolute attributes are amenable to all operations to which ratio attributes are.

Compound attributes have been introduced to concisely and simply describe objects or their components by attributes that are applicable specifically to these objects or components (Michalski, 2004; Wojtusiak and Michalski, 2006). Values of compound attributes are internal conjunctions of values of their constituent attributes. For example, attributes applicable to describing a house can be its type and size, the number of bedrooms, the number of baths, the size of the backyard, etc. To describe individual bedrooms, other attributes are used, e.g., the dimensions of each bedroom, the number of windows, the closet size, etc. In this case, the attributes related only to the bedrooms are combined into a compound attribute called "Bedroom." Using such an attribute, a bedroom can be described by an attributional calculus condition: [Bedroom1 = 20' x 13' & 3 windows & large closet]. Such a statement closely corresponds to a natural language description.

Although compound attributes allow one to describe parts of an object by different sets of attributes, and not the relations between the parts (unless each relation is represented by an attribute), they are useful and sufficient for many practical problems. Their advantages are that operations on them are easy to implement and interpret. Learning descriptions with such attributes is computationally simpler to implement than full-fledged relational descriptions employed, e.g., in inductive logic programming or in the INDUCE method (e.g., Larson and Michalski, 1977). Compound attributes increase descriptions' interpretability because they facilitate expressing knowledge in forms closely related to those used in natural language. Compound attributes are also a useful addition to structured attributes in implementing inductive learning.

In addition to regular values specified in the attribute domain, $\mathcal{AC}$ assumes that every domain also contains three *meta-values* (Michalski, 2004; Michalski and Wojtusiak, 2005). These values represent possible responses to questions requesting the attribute value for a given entity, specifically: "don't know" (denoted by "?"), "not applicable" (denoted by "NA"), and "irrelevant" (denoted by "*"). The meta-value "don't know" is given to an attribute whose value for a given entity exists but is unknown for whatever reason; for example, it has not been measured or has not been recorded in the database. The "NA" value is given to an attribute that is not applicable to a given entity. For example, the attribute "the number of pages" is not applicable to a chair in the library, but is applicable to a book. Finally, the "*" value is assigned to an attribute that can be considered irrelevant to the problem at hand. For example, the attribute "shoe size" can be viewed irrelevant to the problem of learning people's education level from their other attributes. While the value "?" represents lack of knowledge, values "NA" and "*" constitute domain knowledge that is communicated to the system by an expert.

Because attribute type indicates which operators are applicable to the attribute, the type is useful in conducting inductive inference. Inductive generalization rules applicable to statements with attributes of different types are described in (Michalski, 1983). By applying generalization rules according to the attribute type, an inductive generalization process can produce more plausible and understandable generalizations, avoid implausible generalizations, and can also be more efficient. As mentioned earlier, the types and attribute domains are parts of the background knowledge for the given inductive generalization problem. While attribute domains can be inferred from the data (at least approximately), determining attribute types is a more difficult problem, to be handled by a domain expert.

3 SYNTACTIC ATTRIBUTE TYPES

To represent and reason with semantic attribute types efficiently, syntactic attribute types have been introduced to AQ learning. A syntactic type depends not only the semantic type, but also on the size of attribute domain and its other properties. Syntactic types are specified in the user definition of the problem to the learning program. Table 3 lists syntactic attribute types defined in the AQ21 learning program.

Table 3: Syntactic Attribute Types.

- *discrete nominal,* which is used to represent the nominal semantic type.
- *discrete structured*, which is used to represent the structured (hierarchical) attribute type.
- *structured linear* and *structured nominal* subtypes reflect total ordering and no ordering of nodes at the same level of the domain hierarchy, respectively.
- *discrete cyclic,* which is equivalent to the cyclic semantic type.
- *continuous interval,* which is equivalent to the interval semantic type.
- *discretized interval,* which is semantically equivalent to the interval type, but discretized into a finite number of points. Operations applicable to the interval type are executed on discretized interval attributes in the following way: values in discretized examples are first undiscretized to get continuous values in the intervals then the operations are executed. Results are discretized back into the discrete form.
- *continuous ratio,* which is equivalent to the ratio semantic type.
- *discretized ratio,* which is semantically equivalent to the ratio type, but discretized into a finite number of ranges. As in the case of discretized interval type, an analogous three-step method is applied to attributes of this type.
- *compound*, which is equivalent to the compound semantic type. In the AQ21 learning system, compound attributes are projected into constituent attributes for the purpose of learning. Once rules are learned, they are translated back, and displayed in the rules as compound attributes (Michalski 2004; Wojtusiak and Michalski, 2006).
- *set-valued,* which is equivalent to the set-valued semantic type.

Because the AQ21 program operates on syntactic types, their appropriate specification is useful for an effective execution of the program, in particular, for performing such operations as star generation and post-optimization of the learned rules (e.g., Michalski and Kaufman, 2001). Because the specification of syntactic types of attributes may be burdensome for a program user, a method will be developed for their automatic determination, based on the training set.

As described in next section, AQ21 uses a bitstring representation for implementing all discrete attributes, and ranges (defined by pairs of numbers) for continuous attributes. Reasoning with so-represented discrete attributes can be implemented very efficiently. However, because the amount of memory needed to represent discrete attributes (including discretized numerical attributes) grows linearly with their domain size, it is sometimes more efficient to use purely continuous attributes instead. For example, directly reasoning with a Continuous Ratio attribute may be more computationally efficient than with its Discretized Ratio version that has a large number of discrete values. It may be both more desirable and efficient to use discretized forms when they consist only of a few ranges; their higher abstraction level (lower precision) leads to a better generalization and helps avoiding overfitting.

4 PHYSICAL ATTRIBUTE REPRESENTATION

4.1 Bitstring Representation

All discrete attributes, except set-valued, are represented in AQ learning by bitstrings. In such a representation, each value of an attribute is represented by one bit. An additional bit is used to represent the meta-value "Don't know" ("?"). For example, Figure 2 illustrates a bitstring representation for attribute "Color" with five possible values. The presence of a value is indicated by a "1" in the position corresponding to this value and its absence by a "0".

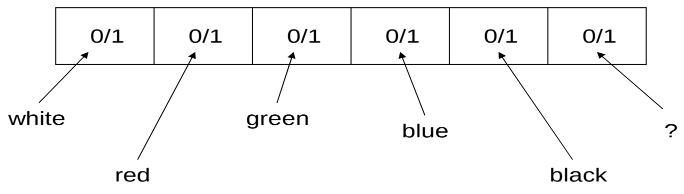

Figure 2: A bitstring representation of attribute "Color" with five possible values.

Using such a representation, single values of discrete attributes in the concept examples, as well as subsets of values defined in attributional conditions of the rules, can be easily represented. For example, the attributional condition [color=*red*] is encoded by a bit vector <010000>, the condition [color= *red v green v blue*] is encoded by <011100>, and (color=?) by <000001>.

In the case of nominal attributes (such as "Color"), the order of values is irrelevant. In the case of attributes with totally ordered domains (e.g., ordinal, cyclic, discretized ratio), bits representing values are ordered as in the attribute domain. Structured attributes are also represented by bitstrings, but their representation includes an additional tree structure representing relationships among values (see Section 4.2). Continuous attributes are represented differently (see Section 4.3).

As one can see, a bitstring representation allows the program to represent both single attribute values and internal disjunctions of values in the same bitstring. The bitstring representation was implemented in the very first implementation of AQ learning in the early 1970s, and in most subsequent versions, including AQ15 (Michalski et al., 1986) and all of its successor AQ programs.

4.2 Representation of Structured Attributes

Domains of structured (hierarchical) attributes are hierarchies. To represent such domains, AQ21 stores bitstrings representing all of their possible values and hierarchy trees representing relationships between values. To illustrate this representation, let us consider the hierarchy presented in Figure 1. The domain of attribute "shape" has sixteen values: *curved*, *circle*, *ellipse*, *triangle*, *right*, *obtuse*, *acute*, *quadrangle*, *square*, *rhombus*, *rectangle*, *irregular*, *polygon*, *pentagon*, *multilateral*, and *hexagon*, organized into a hierarchical structure. All these values are represented by a bitstring consisting of seventeen bits (again, one additional bit represents the "Don't know" meta-value). Relations between values are represented in a double-linked tree, in which each node keeps one link to its parent and one link to a list of offsprings. Such a representation facilitates efficient application of generalization and specialization operators to structured attributes.

4.3 Representation using Ranges of Real Values

Bitstrings are used to represent attributes with finite and small domains. In the real world, there is often the need to use real-valued numerical attributes with the precision represented in the training data. In AQ learning, such attributes or attributes derived from them through constructive induction (e.g. Bloedorn and Michalski, 1998) are encoded using ranges of real values.

A range of values is specified by its lower and upper bound. For example, let "Length" be a ratio attribute with possible values ranging from 0 to 1000. The datapoint "Length=23.456" is encoded in the training data or in a rule as the pair (23.456; 23.456), with both the upper and lower bounds set to the same value. An attributional condition [Length=2.3..29.11] is encoded as (2.3; 29.11).

4.4 Representation of Compound Attributes

Domains of compound attributes are defined as the Cartesian product of the domains of their constituent attributes. Thus, the compound attributes are represented using bitstrings and ranges defined by those domains. In addition, each compound attribute keeps a list of its constituent attributes.

4.5 Representation of "Irrelevant" and "Not-applicable" Meta-values

Meta-values "Irrelevant" ("*") and "Not-applicable" ("N/A") are represented in bitstrings by special combinations of bits. The "*Irrelevant*" meta-value is represented by setting all value bits to one (essentially signifying that all values are acceptable). The "Not Applicable" meta-value is represented by setting all bits to zero (essentially signifying an impossible condition).

To encode meta-values in the range representation, special combinations of upper and lower bound are used: "Don't know," is encoded by the range ($+\infty$; $+\infty$), "Irrelevant" by ($-\infty$; $+\infty$), and

"*Not applicable*" by (+∞; -∞), where ∞ denotes the largest real value representable with a given precision on the host computer.

For further details of the representation of meta-values, and on the method of reasoning with them in AQ learning, see (Michalski and Wojtusiak, 2005).

5 MAPPINGS BETWEEN SEMANTIC AND SYTACTIC TYPES

Relationships between semantic and syntactic attributes and their physical representation are presented graphically in Figure 3. The top two levels in the figure refer to semantic types. These types are grouped into two categories, symbolic and numeric (or qualitative and quantitative). A box between them refers to compound attributes. In order to avoid many connections, the compound attribute type is not connected to other attribute types, because its constituent attributes may be of any type.

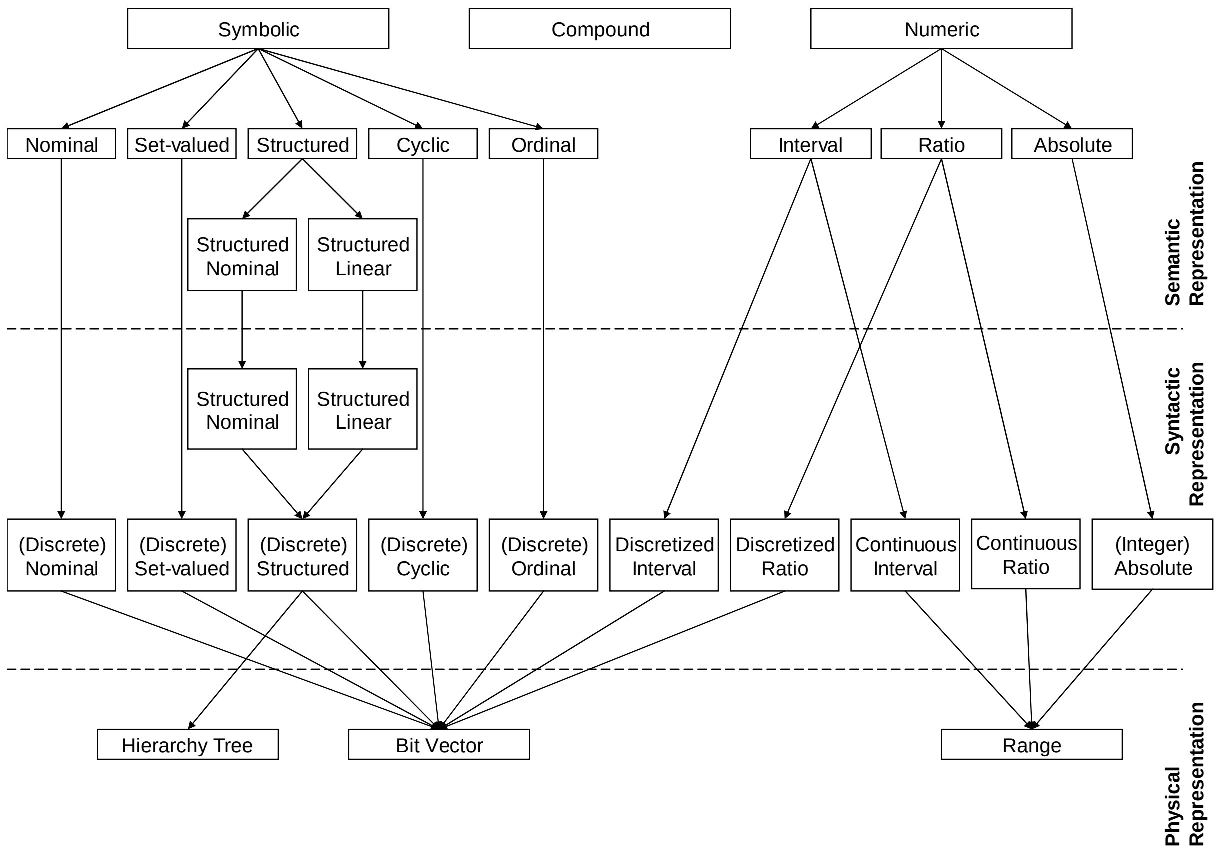

Figure 3: Mappings between semantic and syntactic attribute types, and the data structures used for their physical representation.

The middle layer of the diagram represents syntactic types, as used in AQ21 system. They are used to represent symbolic semantic attribute types. In most cases, there is a one-to-one correspondence between semantic and syntactic types. For example, a *nominal* attributes are always implemented by the *discrete nominal* syntactic type. In some cases, specifically, in the case of structured, interval and ratio semantic types, a semantic type may be represented using one of two different syntactic types. For example *interval* and *ratio* semantic types may be represented by *discretized interval* and *ratio*, or *continuous interval* or *continuous ratio* syntactic types, depending on whether the attribute is discretized or not.

The lowest level of the diagram shows three possible physical representations, *bitstrings*, *hierarchy trees*, and *ranges*. All discrete syntactic attribute types are physically represented by bitstrings. Structured attributes require an additional physical representation to keep information about relations between their values. This is done using *hierarchy trees*. All continuous attributes are represented by ranges (pairs of values).

6 OTHER ATTRIBUTE TYPES

Attribute types provide the learning program with background knowledge useful for performing plausible inductive generalization. In addition to the attribute types presented in this paper, to facilitate induction in specific application domains, it may be useful to introduce additional attribute types, and define generalization operators appropriate for them. For example, one may introduce the attribute type "angle" for applications involving geometrical concepts. This would be a cyclic continuous attribute, whose domain is the set of real numbers between 0 and 360 (degrees), and 360 is equivalent to 0. Operators applicable to attributes of this type include addition and subtraction, as well as all trigonometric functions.

Examples of other attribute types are "date" (of decision, of receiving an ordered product, etc.), "birth date", "age", and "arithmetic equation".

7 RELATED RESEARCH

Although Stevens' attribute types are widely recognized in psychology, most learning programs do not have mechanisms for appropriately reasoning with all of them, not to mention reasoning with the larger set of types described in this paper. Many programs, for example, C4.5 (Quinlan, 1993), RIPPER (Cohen, 1995), and CN2 (Clark and Nibblett, 1989), recognize only nominal (discrete) and continuous (for all numeric types) attributes. The program Weka, which includes several learning methods (Witten and Frank, 1999) additionally supports the type "date". To the authors' best knowledge, the only integrated multi-operator system that supports a wider set of attribute types are INLEN (e.g. Michalski et al., 1992), and its successor VINLEN (e.g. Kaufman and Michalski, 2003; Kaufman et al., 2006).

Structured attributes were first implemented in the INDUCE relational learning program, (e.g. Larson and Michalski, 1977). A detailed study of their use in AQ learning was presented by Kaufman and Michalski (1996). In the literature, structured attributes are now commonly used, and often referred to as hierarchical attributes. They are used in several machine learning and data mining systems, for example, in SAMUEL (Grefenstette, 1992), and in the commercial database Oracle. The importance of this attribute type was recognized by workshop on dealing with structured data in machine learning and statistics at the European Conference on Machine Learning, ECML-2000.

In data mining, authors often recognize nominal, categorical, ordinal, interval, and ratio attribute types. The distinction between nominal and categorical types is that the former type serves the role of a unique identifier, while the later can be, in many cases, used to name groups of objects (e.g. Pyle, 1999). In Attributional Calculus and in AQ learning, no distinction is made between nominal and categorical types, because the difference has no influence on the reasoning process.

Compound attributes, introduced in Attributional Calculus, facilitate learning descriptions of objects whose parts are described by different sets of attributes. Learning descriptions with compound attributes is related to learning relational descriptions, which characterize objects in terms of properties of their parts and relations between the parts (e.g., Larson and Michalski, 1977; Dietterich and Michalski, 1981; Lavrac and Dzeroski, 1994). Compound attributes are also related to the currently very active area of *Probabilistic Relational Learning* that combines statistical and relational learning (PRMs; e.g. Dey and Sarkar, 1996; Getoor et. al., 2001). These methods concern the more general problem of learning relational descriptions, but are less efficient, and do not address the problems of natural induction that specifically stress the comprehensibility and human-orientation of the knowledge to be learned.

8 SUMMARY

To increase the interpretability, predictive accuracy, and efficiency of inductive learning, the AQ methodology for natural induction recognizes a wide range of attribute types, and makes a distinction between semantic and syntactic types. Semantic types govern the process of plausible inductive generalization at the conceptual level, while syntactic types take into consideration physical properties of the attribute domains in order to efficiently implement semantic types. All attribute types are represented at the physical level using data structures: a bitstring, a pair of values (defining a range), and a tree structure.

The presented attribute types are frequently used by people in describing real world objects. This feature therefore reflects the spirit of natural induction implemented in AQ learning.

REFERENCES

Bloedorn, E. and Michalski, R. S., “Data-Driven Constructive Induction,” *IEEE Intelligent Systems, Special issue on Feature Transformation and Subset Selection*, pp. 30-37, March/April, 1998.

Clark, P. and Niblett, T., “The CN2 Induction Algorithm,” *Machine Learning* 3: pp. 261-289. 1989.

Cohen, W., “Fast Effective Rule Induction,” *Proceedings of the 12th International Conference on Machine Learning*. 1995.

Dey, D. and Sarkar, S., “A Probabilistic Relational Model and Algebra,” *ACM Transactions on Database Systems (TODS)*, 21:3, 1996.

Dietterich, T. G. and Michalski, R.S., “Inductive Learning of Structural Descriptions: Evaluation Criteria and Comparative Review of Selected Methods,” *Artificial Intelligence*, 16:3, pp. 257-294, July, 1981.

Grefenstette, J.J., “The Evolution of Strategies for Multiagent Environments,” *Adaptive Behavior*, 1, 1, 65-90, 1992.

Getoor, L., Friedman, N., Koller, D., and Taskar, B., “Probabilistic Models of Relational Structure,” *International Conference on Machine Learning*, Williamstown, MA, June, 2001.

Kaufman, K. and Michalski, R.S., “A Method for Reasoning with Structured and Continuous Attributes in the INLEN-2 Multistrategy Knowledge Discovery System,” *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (KDD-96), Portland, OR, pp. 232-237, August, 1996.

Kaufman, K. and Michalski, R.S., “The Development of the Inductive Database System VINLEN: A Review of Current Research,” *International Intelligent Information Processing and Web Mining Conference*, Zakopane, Poland, 2003.

Kaufman, K., Michalski, R.S., Pietrzykowski, J. and Wojtusiak, J., “An Integrated Multi-task Inductive Database and Decision Support System VINLEN: An initial implementation and first results ,” *Fifth International Workshop on Knowledge Discovery in Inductive Databases, KDID'06, in conjunction with ECML/PKDD*, Berlin, Germany, September, 2006.

Larson, J. and Michalski, R.S., “Inductive Inference of VL Decision Rules,” *Invited paper for the Workshop in Pattern-Directed Inference Systems*, Hawaii, and published in SIGART Newsletter, ACM, No. 63, pp. 38-44, June 1977, May 23-27, 1977.

Lavrac, N. and Dzeroski, S., *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood, 1994.

Michalski R.S., “Pattern Recognition as Knowledge-Guided Computer Induction,” *Report No. 927*, Department of Computer Science, University of Illinois, Urbana, IL, May, 1978.

Michalski R.S., “A Theory and Methodology of Inductive Learning,” in Michalski, R.S., Mitchell, T. and Carbonell, J. (eds.), *Machine Learning: An Artificial Intelligence Approach*, Tioga Publishing Co., Palo Alto, CA, pp. 83-134, 1983.

Michalski, R.S., “Concept Learning and Natural Induction,” *Lecture Notes on Machine Learning and Applications, Advanced Course on Artificial Intelligence*, Chania, Greece, July, 1999.

Michalski, R.S., “ATTRIBUTIONAL CALCULUS: A Logic and Representation Language for Natural Induction,” *Reports of the Machine Learning and Inference Laboratory*, MLI 04-2, George Mason University. 2004.

Michalski, R.S., Mozetic, I., Hong, J. and Lavrac, N., “The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains,” *Proceedings of the National Conference on Artificial Intelligence*, AAAI, Philadelphia, August 11-15, 1986.

Michalski, R.S., Kerschberg, L., Kaufman, K. and Ribeiro, J., “Mining For Knowledge in Databases: The INLEN Architecture, Initial Implementation and First Results,” *Intelligent Information Systems: Integrating Artificial Intelligence and Database Technologies*, Vol. 1, No. 1, pp. 85-113, August ,1992.

Michalski, R. S. and Kaufman, K., "Learning Patterns in Noisy Data: The AQ Approach," *Machine Learning and its Applications*, G. Paliouras, V. Karkaletsis and C. Spyropoulos (Eds.), pp. 22-38, Springer-Verlag, 2001.

Michalski, R.S. and Wojtusiak, J., “Reasoning with Meta-values in AQ Learning,” *Reports of the Machine Learning and Inference Laboratory*, MLI 05-1, George Mason University, Fairfax, VA, June, 2005.

Pyle, D., *Data Preparation for Data Mining*, Morgan Kaufmann, 1999.

Quinlan, J.R. *C4.5 Systems for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

Stevens, S.S., “On the Theory of Scales of Measurement,” *Science*, 103, pp. 677-680, 1946.

Witten, I.H. and Frank, E., *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufmann, 1999.

Wojtusiak, J. and Michalski, R.S., “The Use of Compound Attributes in AQ Learning,” *Proceedings of the Intelligent Information Processing and Web Mining Conference*, IIPWM’06, Ustron, June 19-22, 2006.

Wojtusiak, J., Michalski, R.S., Kaufman, K. and Pietrzykowski, J., “The AQ21 Natural Induction Program for Pattern Discovery: Initial Version and its Novel Features,” *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence*, Washington D.C., November, 2006.

Vellman, P.F. and Wilkinson, L., “Nominal, Ordinal, Interval, and Ratio Typologies Are Misleading,” *American Statistician*, 47:1, pp. 65-72, February, 1993.

A publication of the *Machine Learning and Inference Laboratory*
George Mason University
Fairfax, VA 22030-4444 U.S.A.
http://www.mli.gmu.edu

Editor: R. S. Michalski
Assistant Editor: Janusz Wojtusiak