Reports of Machine Learning and Inference Laboratory
June 23, 2001

Attributional Calculus

A Representation System and Logic for Deriving Human Knowledge from Computer Data

Ryszard Michalski*
Machine Learning and Inference Laboratory
School of Computational Sciences
George Mason University
Fairfax, VA

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

Abstract

Attributional calculus is a typed logic system that combines elements of propositional logic, predicate logic, and multiple-valued logic. It was developed with the main intention to support *natural induction*, by which we mean an inductive inference that generates knowledge in the forms "natural" to people, such as natural language-style descriptions or graphical visualizations, and by that simple to interpret and easy to relate to human knowledge. To this end, attributional calculus employs new descriptive constructs and operators that can significantly simplify descriptions that, if represented using standard logic operators, would be complex and opaque. Sentences in attributional calculus can be interpreted as binary logic expressions (*crisp interpretation*), or as multiple-valued or continuously-valued logic expressions (*flexible interpretation*). Attributional calculus stems from variable-valued logic (VL_1), and its various subsets have been be implemented in the AQ-type learning programs. Conventional decision rules and association rules employed in data mining can be viewed as special cases of attributional rules.

Keywords: Inductive inference, machine learning, data mining, knowledge discovery, natural induction, propositional calculus, predicate logic, many-valued logic, attributional calculus.

1 Motivation

The explosive growth of databases and information systems creates a strong need for methods that can derive from them useful knowledge. In many application areas, such as economy, medicine, agriculture, management, sciences, and others, computer-created knowledge has to be not only accurate but also easy to interpret and understand by humans. The latter condition stems from the fact that human experts in these areas are highly reluctant to employ in their decision-making any such knowledge blindly, without understanding it and relating it to their other knowledge.

To emphasize this issue, we have introduced the concept of *natural induction*, by which is meant a process of creating inductive hypotheses in the forms that are "natural" to people, and by that easy to interpret and relate to other human knowledge.

What is natural to people is, of course, subjective and context-dependent. Therefore, the above goal is approximated by requiring that induced formal hypotheses be syntactically and semantically similar to logically equivalent natural language descriptions, and/or represented in easy to interpret visual forms. The former requirement reflects the fact that natural language is

the basis for human communication and knowledge exchange. Various areas of science added special concepts and notations to natural language (e.g., mathematical symbols and expressions, special notation for logic operators and inference rules, etc.), but natural language remains the foundation for human knowledge representation. Because some pictorial forms of representing knowledge are also natural to people (graphs, diagrams, maps, flow-charts, schemas, graphical representations of chemical compounds, etc.), we have developed methods for visualizing attributional calculus expressions generated by inductive learning programs. Such methods include *diagrammatic visualization* (Michalski, 1978; Zhang, 1997) and *attributional graphs or association graphs* (Michalski and Kaufman, 2000).

To develop a natural induction system, one needs an adequate knowledge representation language. Motivated by this goal, we have been working on various formal languages that would support natural induction and at the same time would be relatively easy to implement. This paper describes a recent result of these efforts, *attributional calculus*, which is a typed logic system combining elements of propositional logic, predicate logic, and multiple-valued logic. Expressions in attributional calculus are descriptions that characterize entities of interest in terms of discrete or continuous attributes. Attributional calculus can thus be viewed as a representation system for building attribute-based descriptions of any entities or groups of entities, and a logic formalizing reasoning in a subset of natural language about such entities.

As mentioned above, the main motivation for natural induction is to assure that descriptions induced from data be easy to understand and interpret by people. Such a requirement has been known as the *postulate of comprehensibility* (Michalski, 1983), and is important or even crucial for a wide range of applications of machine learning, data mining and knowledge discovery.

One may be tempted to propose an approach to natural induction in which inductive inference is conducted in a language alien to human cognition, but convenient computationally, and require that only results of inference be translated to "natural", human-oriented forms. Unfortunately, such an approach is doomed to failure. This so because the representation language in which system conducts inductive inference (also called a representational bias) has a direct influence on the kind of generalizations it will create and select among a plethora of possible hypotheses that usually can be induced from a given dataset. Inductive inference is, by its very nature, an under-constrained problem, and the same data may beget many different generalizations that are (fully or approximately) consistent and complete with regard to the given data.

Although the above argument appears similar to the Sapir-Whorf hypothesis (that the language we speak strongly influences our thinking; Whorf 1956), it has here a different meaning. The process of generating inductive hypotheses typically involves many steps at which intermediate hypotheses (generalizations) are created and evaluated. At each of these steps, an inductive reasoner must made a decision as to which of the alternative generalizations should be explored further and which should be discarded. Generalizations expressed in one representation language may score high on a given criterion of hypothesis quality (e.g., some form of simplicity), but score low when expressed in another language. Therefore, the system has no way to know which currently evaluated hypotheses may eventually lead to cognitively simple representations, unless the language in which hypotheses are generated is similar to the ultimate target language, that is, in this case, natural language. This argument has been supported by experiments reported in (Michalski and Wnek, 1992), which show that methods using opaque knowledge representations, such as artificial neural nets or classifiers using a genetic algorithm were unable to correctly learn target concepts (representable by very simple natural language descriptions), even when the training set contained all possible positive examples.

Consequently, an approach based on a *post factum* translation of generated hypotheses to logically equivalent natural forms cannot accomplish aims of natural induction. To accomplish them, the process of inductive inference must impose cognitive constraints on the inference process itself, that is, must be carried out from the outset in a cognitively-oriented description

language.

2 Attributional Calculus vs. Other Logic Systems

Attributional calculus has three forms: *core form*, *extended form*, and *annotated form*. The core form can be characterized as a propositional logic in which propositional letters (standing for a sentence or its negation) are replaced by *attributional conditions* (also called *unit sentences* or *selectors*). Attributional conditions correspond to simple natural language sentences that describe groups of entities in terms of one, or at most few attributes.

The extended form adds to the core form additional constructs, such as *meta-attributes* (which allow one to represent a wide class of quantified expressions), and the *exception operator*. The annotated form annotates core attributional expressions with *merit parameters* that provide a numerical characterization.

There are two *interpretation schemas* for attributional calculus sentences: *crisp* and *flexible*. The crisp schema treats attributional sentences as binary logic expressions, they evaluate either to truth or falsity. The flexible interpretation schema treats attributional sentences as multi-valued logic expressions; they evaluate to a degree of truth that can be either discrete or continuous. In the following, we will assume the crisp interpretation schema. The latter part of the paper will discuss attributional calculus with a flexible interpretation schema.

In propositional logic, declarative natural language sentences are represented by single literals (propositional letters or its negations). Such a representation does not allow one to directly represent the structure of sentences. If there is a relation between components of two natural language sentences, information about it is lost when sentences are abstracted into single literals. Attributional calculus allows one to represent simply and directly some common relations involving attributes of objects. Unlike propositional calculus and first order predicate logic, it also recognizes different types of attributes. Attribute types are used to direct the inductive inference process accordingly to generalization rules associated with each type (Michalski, 1983).

Due to the use of additional operators (internal logic operations, range, exception), and *meta-attributes*, attributional calculus has a significantly greater pragmatic and formal representational power than propositional logic. The greater pragmatic power means that it can represent some functions in a more compact and simpler way than propositional calculus. The greater formal power means that it can represent some functions that cannot be represented in propositional calculus (such as discrete-output functions with continuous arguments). If all attributes are binary and meta-attributes are not allowed, attributional calculus reduces to propositional calculus.

For some classes of functions, attributional calculus has also greater pragmatic representation power than first order predicate logic. As its name indicates, it is suitable, for representing attribute-based descriptions, but not structural descriptions, like first order predicate logic. Its most important features are that it is easily and directly interpretable in natural language, and thus facilitates natural induction, that it can be efficiently implemented, and admits a flexible (multi-valued) interpretation.

3. Explanation of Basic Concepts

3.1 Universe of Discourse

Attributional calculus is a formal language and an inference system for supporting inductive inference about any entities (objects, processes, behaviors, etc.) that can be adequately described in terms of their attributes (zero-argument functions). The set of all entities to be considered in the inference process is called the *universe of discourse.* For example, the universe of discourse may be the set of patients, plants, behaviors, minerals, songs, stocks, goods in a department

store, etc.

It is assumed that information about the entities from the universe of discourse is in the form of values of attributes given in advance. Attributional calculus sentences are interpreted by applying them to entities from the universe of discourse.

3.2 Attributes and Their Types

An attribute is a mapping from a set of entities to the domain (the set of legal values) of the attribute. Such a mapping is done by a measuring device, human judgment, or some computational process. The structure of the attribute domain defines the *type* of the attribute. The type of attribute determines the class of operations that can be meaningfully performed on the attributes of that type.

The attribute types are related to the well-known measurement scales. In addition to the types corresponding to standard measurement scales (nominal, ordinal, interval, ratio and absolute), attributional calculus also recognizes cyclic, structured, and set-valued attribute types. Below is a list of types defined in attributional calculus:

nominal, if the attribute domain is an unordered set (e. g., "blood type" or "person's name").

linear, if its domain is a totally ordered set (e.g., student's grade, temperature of an object, length, height, the number of desks in an office, etc.). Linear attributes can be in turn classified into rank, interval, ratio, and absolute, corresponding to their measurement scale.

cyclic, if its domain is a cyclically ordered set (e.g., days of the week, signs of the zodiac, time zones, etc.).

structured, if its domain is a hierarchically ordered set (e.g., a type- or generalization-hierarchy of plants, animals, geometrical shapes, diseases, etc.).

set-valued, if its domain is the power set of a *base set*; attribute values are thus subsets of the base (e.g., if the base set is the set of all products in a store, a set-valued attribute could be "products purchased by customer A in that store").

The type of an attribute helps to guide inductive generalization processes involving this attribute. Specifically, different rules of inductive generalization apply to statements with attributes of different type (Michalski, 1983).

In addition to legal values, every attribute domain is assumed to contain three *meta-values*. These are not regular attribute values, but possible responses to questions requesting an attribute value, namely: "do not know" (denoted by "?"), not applicable (denoted by "NA"), and irrelevant (or "don't care," denoted by "*").

The meta-value "NA" is given to an attribute that is not applicable to a given object. For example, the attribute "the number of pages" is not applicable to a flower, but applicable to a book. The attribute "the person's eye color" may be declared irrelevant to the problem of determining that person's education level.

Attributional calculus builds upon the *variable-valued logic one* (VL1), a multiple-valued logic system introduced by Michalski (1972, 1994).

3.3 Events and Representation Space

Every entity from the universe of discourse can be described by values of attributes applied to that entity. A vector of attribute-values for an entity is called an *event.* The *representation space* for a given universe of discourse is the set of all possible events employing given attributes. If attributes from ***A*** are used to describe entities from a universe of discourse, then it is said that

attributes in ***A*** span the representation space for this universe of discourse.

4 Definition of Attributional Calculus

4. 1 Attributional conditions

A basic representational unit of attributional calculus is an *aAttribution condition* (a.k.a. *selector)* which roughly corresponds to a simple sentence in natural language. A selector relates one or more attributes to their values or other attributes. A general form of a selector is:

[L rel R],

where

L (*left side*) is an expression specifying an attribute, or a group of attributes with the same domain. Attributes from the group are joined by "&" or "v", called *internal conjunction* and *disjunction*, respectively. ***L*** can also be one of the meta-attributes: *count*, *max*, *min*, and *ave*.

R (*right side*) is an expression specifying a value or a subset values from the domain of the attribute/s in ***L***. If the subset contains values of a nominal attributes, they joined by the symbol "v" (called *internal disjunction*); the subset contains consecutive values of an linear attribute, they are represented by joining the extreme values by operator"..", called *range*. ***R*** can also be an attribute with the same domain as the attribute or attributes in ***L***.

rel is a relational symbol from the set: {=, ≠, >, ≥, <, ≤}. Relation operators {=, ≠} apply to all types of attributes. Relations {>, ≥, <, ≤} apply only to linear attributes.

Brackets [ ] can be omitted if their omission leads to no confusion. If brackets are used, the conjunction of two selectors can be written as their concatenation.

Attributional conditions are interpreted in the context of an entity (object, situation, etc.) they are applied to. They can have a prepositional (or binary) interpretation, or multi-valued interpretation. In the prepositional interpretation, ***[L rel R]*** is said to be *satisfied* by the entity (or evaluates to *true*), if **L** is in relation ***rel*** to ***R*** for that entity. In the multi-valued interpretation of an attributional condition, ***[L rel R]*** is said to be *satisfied* or *matched* by the entity to a degree α (or, simply, to evaluate to α), if for that object **L** is in relation ***rel*** to ***R*** according to a given *flexible evaluation scheme*.

An attributional condition is called *elementary*, if its left side, ***L,*** is a single attribute, the relational symbol is one of { = > ≥ < ≤ }, and the right side, ***R*** is a single value; otherwise, it is called *composite*. If an attribute, x, is binary, the condition [x = 1] can written simply as literal x, and [x= 0] as literal ~x. Thus, if attributes are binary, attributional conditions reduce to propositional literals.

Examples of attributional conditions with their natural language interpretation are presented below.

Elementary attributional conditions

[x_1 = 0]	(The value x_1 is 0)
[color= red]	(The color is red)
[length < 5"]	(The length is smaller than 5 inches)
[temperature ≥ 32^0C]	(The temperature is greater or equal 32^0C)
[price= ?]	(The price is unknown)

Composite conditions

[color = blue v red]	(the color is blue or red)

[length= 4..12] (the length is between 4 and 12, inclusively)

[color ≠ green] (the color is not green)
[height > width] (the height is greater than weight)

[height v width ≤ 3m] (the height or width is smaller than 3m)

[height & width ≥ 7cm] (the height and width is greater than or equal 7cm)
[height & width < length] (both height and width are smaller than length)
[count(x_1, x_2, x_3, x_4: EQ 3)=1] (there is exactly one attribute in the set X that has value 1)

The last composite condition involves a meta-attribute "count" (see next section).

Note: "v" or "&" when applied to non-binary attributes or their values represent internal disjunction or conjunction, respectively.

Note that attributional conditions are easy to interpret and easy to translate to an equivalent natural language statement. The can express any condition on a subset of attribute values, and be easily generalized or specialized (by applying an *internal disjunction* with nominal attributes, the *range* or *relation* operator with linear attributes, or the *climbing-generalization-tree* operator with structured attributes (see sec. on *Inductive Inference*).

4.2 Meta-attributes

The extended form of attributional calculus also recognizes *meta-attributes*, which are forms of quantifiers over attributes. The following meta-attributes are defined: *count*, *max*, *min*, and *ave*. The count attribute applies to attributes of any type, but the domains of these attributes must be the same. The other meta-attributes apply only to sets of linear attributes. Attributional conditions with meta-attributes are composite.

The meta-attribute count applies to all types of attributes, and is in the form:

count{X: REL V}

where X stands for a set of attributes with the same domain, REL stands for a relation from the set: {EQ, NEQ, GT, EGT, ST, EST}, and V is one or more values from the domain of elements in X.

The count meta-attribute determines the exact number of items (attributes, sentences) in X whose value is in relation REL to one of the values defined by V. The symbol EQ means equal, NEQ – not equal, GT – greater than, EGT—equal or greater than, ST- smaller than, and EST-equal or smaller than. For example, the meta-attribute:

count(x_1, x_2, x_3, x_5: EQ 1)

can be interpreted as "the number of attributes in {x_1, x_2, x_3, x_5} that take value 1 in a description of the event to which it is applied."

The domain of the count attribute is thus: {0,1,2,…,|X|}. If the count takes value |X|, the cardinality of X, then one can alternatively say that the value of count{X: REL V} is *all*. As regular attributes, the count meta-attribute can also take meta-values "?" (don't-know), and "NA" (not applicable---when X is not a set of attributes). The count meta-attribute is used in the same way as any other attributes to create attributional conditions. Attributional conditions with a count attribute are composite.

In the above definition of the meta-attribute count, X is a set of (regular) attributes. This form of

the count meta-attribute is called the *first-order count* attribute. The meta-attribute count in which X is the set of original attributes is used in attributional conditions in the same way as any other attribute.

In the count meta-attribute, X can be not only a set of attributes, but also a set of attributional sentences. In the latter case, the count meta-attribute counts the number of sentences that are true or false (in the strict evaluation schema), or the number of sentences whose evaluation is in relation REL to V (in the flexible evaluation schema).

It is easy to see that the count attribute supports an existential and universal quantification over the elements in X. Assume, for example, that $X = \{x_1, x_2, x_3, x_4, x_5, x_6\}$, where all x_i have the same domain, and the predicate EQ(x, v) states that attribute x has value v. The existentially quantified expression:

$$\exists\, x \in X,\ \ EQ(x, v)$$

can be written in attributional calculus as:

$$\text{count}(X{:}\ EQ\ v)\ \geq 1$$

and expressed in natural language as "the number of variables in X with value equal v is greater or equal to 1." The universally quantified expression:

$$\forall x \in\ X,\ \ EQ(x, v)$$

can be written in attributional calculus as:

$$\text{count}(X{:}\ EQ\ v) = \text{all}$$

In addition to existential and universal quantification over elements in X, the meta-attribute count allows one to express also various numerical forms of quantification that do not exist in conventional predicate logic. Such expressions include, for example:

count(X: EQ 4 v 5) > 2 (the number of variables in X whose value 4 or 5 is greater than 2),

count(X: EQ 5) = 2..4 (the number of variables in X whose value 5 is between 2 and 4).

count([x=1]&[y>4], [color=red or blue], [z=2..5]: True) = 2 (among attributional sentences :{[x=1]&[y>4], [color=red or blue], [z=2..5]} two and only two are true).

The meta-attribute count is useful in many application domains, for example, in medicine, where it is not uncommon that a specific disease is indicated if the patient has the number of symptoms from a given set greater than some threshold. The count meta-attribute provides an elegant means for expressing the so-called *n-of-m* relations (Sebag, 1999).

It also helps to express simply symmetric Boolean logic functions, which tend to have the most unwieldy expressions in terms of standard logic operators (Michalski, 1969). In general, the count attribute provides a mechanism for expressing simply logic functions whose expressions using standard logic operators would be very long and opaque.
The meta-attributes max, min, and ave are written in the form:

$$\max\{X\},\ \min\{X\} \text{ and } \text{ave}\{X\}$$

and express respectively the maximum, the minimum, and the average value of attributes in X when applied to a given entity. For example, given an event (3,6,4,9,8), max(X)=9, min(X)=3, and ave(X) = 10.

The motivation for introducing meta-attributes to the attributional calculus is that it is very difficult to express their meaning in terms of conventional logic constructs. By introducing them to the language, one can compactly and comprehensibly express a large range of cognitively simple relations that otherwise would be difficult to express. Meta-attributes thus facilitate natural induction.

4.3 Syntax and Semantics of Attributional Calculus

The syntax of attributional calculus is defined by the following rules (WFF means a well-formed expression in attributional calculus):

1. An attributional condition (also called a *selector* or *atomic sentence*) is a WFF. (The semantics is defined as in Sec. 3.4. and 3.5).

2. If *S* is a WFF, then ~*S* is a WFF (If *S* is true, then ~*S* false, and conversely). A negated condition, ~S, can be represented by appropriately changing relation in the condition. E.g., ~ [x = a] is equivalent to [x ≠ a], and ~[x > a] it equivalent to [x ≤ a].

3. If *S1* and *S2* are WFFs, then

S1 & S2 Conjunction: Both *S1* and *S2* are true,

S1 v S2 Disjunction: *S1* or *S2* or both are true,

S1 ⇒ S2 (written also S2 <= S1) Implication: If *S1* is true then *S2* is true, otherwise, S2 is unknown,

S1 ⇔ S2 Equivalence: *S1* and *S2* are both either true or false,

S1 v̲ S2 Exclusive-or: Either *S1* or *S2* is true, but not both),

S1 \ S2 Exception: S1 is true unless S2 is true, in which case S1 is not true. The operator is logically equivalent to Exclusive Or, but is used only when the ratio of the frequencies of S1 being true over S2 being true exceeds the *exception threshold* (a parameter for exception)

are WFFs.

If R is a symbol or a name, and S is an attributional calculus sentence (WFF), then an assignment:

R = S is a WFF. If a symbol is assigned a WFF, then it is by itself also a WFF.

4.4 Cognitive constraints

In additional to the syntax and semantics, attributional calculus includes also constraints on the complexity of its sentences, called *cognitive constraints*. These constraints reflect preferences and limitations imposed on attributional sentences in order to make equivalent natural language expressions appear natural to people, and by that easy to interpret and understand.

The cognitive constraints are defined by parameters limiting the number of repetitions of different operators in one attributional calculus sentence. These parameters include:

- c- conjuction constraint: the maximum number of conjunction operators in a conjunctive statement (e.g., c=5)
- d- internal disjunction constraint: the total number of internal disjunction or conjunction operators in an attributional condition (e.g., 4)
- e- disjunction constraint: the maximum number of disjunction operators in a sentence (e.g., e = 2)
- i implication constraint: the maximum number of implication operators in a sentence (e.g., i =1)
- eq equivalence constraint: the maximum number of equivalence operators in a sentence (e.g., eq =1)
- ex exception constraint: the maximum number of exclusive-or or exception operators in a sentence (e.g., ex =1)

There is also a limit on the sum s= e + i + eq + ex (e.g., s= 1)

Cognitive constraints are imposed when results of inductive inference (generalizations, rules,

patterns, etc.) are to be outputted by the system. If a sentence violates cognitive constraints, then it is called inadmissible expression. Such an expression can replaced by a logically equivalent set of expressions that satisfy the constraints, called *admissible* attributional expressions, or, briefly, *attributional sentences.* Such a transformation is done by assigning literals to parts of the attributional expression that are admissible expressions, and then using literals instead of these parts.

An attributional sentence consists of attributional conditions and/or literals that stand for attributional sentences. An ordered set of attributional sentences is called an *attributional description*, if the first occurrence of some literal in a sentence is followed by its assignment to a corresponding attributional sentence.

For example, assuming c=4, a conjunctive expression A & B & C & D & E & F & G & H, where A,B,.C, H are selectors or literals, can transferred to an attributional description:

<A & B & I & J, I = C & D & E , J = F & G & H>

This description is read: "A and B and I and J, where I is C and D and E, and J is F and G and H."

5 Important Cases of Attributional Sentences

5.1 Attributional Conditions with Meta-attributes

Meta-attributes are used in attributional conditions the same way as any attributes. Below we give various examples of expressions with meta-attributes.

Example 1. The statement" "exactly two variables in {x1, x3, x6, x9} have value greater than 3" can be expressed as:

[count{x1, x3, x6, x9 : GT 3} = 2]

Example 2. The statement: "It two or more conditions from among [x1=2], [x2=4], and [x5 > 1] are true than d > 4" can be expressed as

[count{x1=2, x2=4, x5 > 1} ≥ 2] => [d > 4]

Example 3. The statement:

If two or fewer conditions from among PSA < 2.5, DRE = no,BIO= neg are true, and PS is No, then state is abnormal" can be expressed as:

[Count {PSA < 3.5, DRE = no, BIO= neg} ≤ 2] & [FU=Yes] => [State = abnormal]

This above statement can be expressed alternatively as:

[Count {PSA ≥ 3.5, DRE = yes, BIO= yes} ≥ 1] & [FU=Yes] => [State = abnormal]

The meta-attributes max, min, and ave are in the form:

max{X}, min{X} and ave{X}

and express the maximum, the minimum, and the average value of the attributes in X.

The motivation for introducing meta-attributes to the attributional calculus stems from the fact that it is very difficult to express their meaning in terms of conventional logic constructs. By introducing them to the language, one can simply express a large range of relations involving attributes.

5.2 Attributional Descriptions and Rules

Attributional descriptions are attributional calculus expressions that describe one or more of

entities. and a function mapping a group of entities to a set of decision about the group. A basic description is a *conjunctive description (*also called a *complex*), which is a conjunction of attributional conditions.

Example: **My favorite house**

[Color =white v brown] & [#Bedrms =4..5] & [Backyard=large & private] & [Dist-to-work-H&Dist-to-work-W ≤ 5mil] & [School_quality = good] & [Shops-from-w v Shop-from-h ≤ 3mil] & [LocAreaDrgDlrs=prsn]

where the last condition means: "the location of drug dealers who operated in the local area is prison."

A conjunctive attributional rule (briefly, *an attributional rule)* is a conjunctive description linked by implication to another conjunctive description.

Example: **What to do in a nice weather**?

[Weather=nice] & [Have_partner=yes] => [Drop_CS580] & [Play = tennis]

A statement A => B, where A and B are attributional expressions is called an *attributional rule* (or an attributional implicative statement). In an attributional rule, A is called premise and B is called consequent. If in an implicative statement A => B, premise contains a disjunction of attributional conjunctions, then the attributional can be re-represented by a set of attributional rules. An attributional rule in which consequent has more than one condition is called *multihead attributional rule*, otherwise it is called a single-head attributional rule.

An attributional rule, A => B, in which B specifies a decision is called a *decision rule.* For example: [price=low] & [quality=high] => [decision= buy]

A collection of decision rules with the same decision inconsequent is called a *ruleset.* A ruleset corresponds to disjunctive normal form (a disjunction of conjunctions), and serves as a concept description in AQ attributional learning

A collection of rulesets associated with a single decision attribute (a collection that contains a ruleset for each value of the decision attribute) is called a *ruleset family.*

A collection of ruleset families for one more decision attributes from the same database is called *a knowledge class.*

A knowledge class with associated annotations (definitions of attribute types, rulesets parameters, and related information) and the associated database (e.g., datapoints used in the training set of the knowledge class) is called *knowledge system*

Another reason, perhaps more important, stems from the fact that decision trees and conventional decision rules are knowledge representations of relatively low expressive power. Consequently, to express some cognitively simple descriptions may require a very complex decision tree, or an unreasonably many conventional rules (which use conditions in the form "attribute-value" or "attibute-rel-value", where rel is $\geq$ or $\leq$). Some learned descriptions thus cannot be easy to understand though they may represent a simple concept.

5.3 An Example of an Attributional Rule and Its Implementation

Consider a decision rule expressed in natural language as:

If $x_1 \leq x_2$, $x_3 \neq x_4$, and x_3 is red or blue, then make decision A (6.1)

If variables x_i, i=1,2,3,4, are two-valued, then representing (1) would require a decision tree with 26 nodes and 20 leaves, or 12 conventional decision rules. If variables x_I are five-valued, then representing (1) would require a decision tree with 810 leaves and 190 nodes, or 600

conventional decision rules. The need for such a complex representation of a relatively simple relationship demonstrates a major limitation of these representational formalisms. If the learning system could express the above rule in a form identical or directly related to (1), that is, in a "natural" form, then resulting representation would not only be simpler to manage computationally but also easier to understand and interpret by a user.

This example illustrates a central design criterion for the AQ18 system, which calls for employing a knowledge representation that can express learned knowledge in the form as natural to people as possible. A system satisfying such requirement is called a *natural induction* system (Michalski, 1999). The above knowledge representation must, however, satisfy certain constraints, namely, be syntactically and semantically well-defined and relatively easy to implement. Therefore, it cannot be natural language.

To approach such a goal, AQ18 employs *the attributional calculus*, a highly expressive description language based on variable-valued logic system VL1* (Michalski, 2000). (VL1* can be viewed as a logic system that occupies a place between propositional calculus and predicate calculus.

For example, a VL1* expression of (1) would be:

$$[\text{Decision} = A] \;<=\; [x_1 \le x_2] \;\&\; [x_3 \ne x_4] \;\&\; [x_3 = \text{red v blue}] \qquad (6.2)$$

In order to make the rule (2) more readable, and to provide a user with more information about it, AQ18 would print it in a form even closer to (1), and would add to annotations:

[Decision = A] if [$x_1 \le x_2$: 1998, 966] & [$x_3 \ne x_4$: 80, 19] & [x_3= red or blue: 780, 40]

Evaluations: t=750, u=700, n=14, f=4, q=.9
Ambiguous: (pointers to examples that satisfy the rule but also belong to another class)
Exceptions: (pointers to examples of other classes that satisfy the rule conditions)
Positives covered: (pointers to all positive examples covered by the rule)
Flexibly covered: (pointers to all positive examples flexibly matched)
(3)

where

-- bracketed statements indicate single conditions in the rule,
-- a pair of numbers after ":" in every condition specifies the number of positive and negative examples covered by the condition, respectively
--*Evaluations* is a list of values of various rule characteristics t, u, n, f, q
--t specifies the total number of the positive examples covered by the rule (rule coverage)
--u specifies the number of examples covered only by this rule, and not by any other rule associated with Decision=A
--n denotes the number of negative examples covered by the rule ("negative coverage')
--q denotes the *rule quality* as measured by a rule quality criterion based on rule coverage and *training accuracy* (Kaufman and Michalski, 1999)
--f denotes the number of examples in the training set that are matched *flexibly* (Section 6)
--*Ambiguous is* a list of covered training examples that belong to both positive and negative classes
--*Exceptions* (false positives) is a list of pointers to n negative examples covered by the rule
--*Positives covered* (true positives) is a list of pointers to positive examples crisply covered by the rule.
--*Flexibly covered* is a list of pointers to positive examples that are covered by flexible matching.

To distinguish conventional decision rules (which use conditions in the form attribute=value or attribute-rel-value, where rel is "≤" or "≥") from AQ18's rules (which can use more expressive conditions, see Section 4.4), the former rules are called *elementary* and the latter are called *composite* (Michalski, 2000). As illustrated by the example above, the composite rules may be able to represent some functions much more compactly and understandably than elementary rules.

In AQ18, a result of a learning process is a family of *rulesets*, where each ruleset is a collection of attributional rules that together characterize one decision class (thus a ruleset is equivalent to a *disjunctive normal form* (DNF) description of the class). Below we describe briefly the most important features of AQ18.

5. 4 Expressive Power of Attributional Calculus

Attributional calculus can be used to describe mappings of sets of objects represented in a *finite or infinite representation space* into a discrete set, called the *target set*.

Theorem

Attributional calculus can be used to express any mapping:

$$f: \quad D_1 \times D_2 \times D_3 \times D_n \text{ ----> } D^1 \times D^2 \times D^3 \times \ldots \times D^m \qquad (7.1)$$

where D_1, D_2, D_3, …, D_n are discrete or continuous *input domains*, and D^1, D^2, D^3, …., D^m are discrete *output domains*, and x denotes cartesian product.

Proof

Since output domains are discrete, the product $D^1 \times D^2 \times D^3 \times \ldots \times D^m$ is also a discrete set. Assume then without losing generality that $D_o = D^1 \times D^2 \times D^3 \times \ldots \times D^m$.

Let us now consider two cases:

DI, in which D_1, D_2, D_3, D_n are all discrete sets (discrete input) and CI, in which some or all D_1, D_2, D_3, ,D_n are continuous.

DI case, (1) can be represented as:

$$f: \quad D_1 \times D_2 \times D_3 \times D_n \text{ ----> } D_o \qquad (7.2)$$

For each value of D_o, we can build a ruleset that maps the set in the left-hand-side of (2) into that value (an attributional ruleset).

The union of premises of rules linked to one value of D_o is equivalent to a disjunctive normal form (DNF) describing that value. A DNF can represent any function that maps a combination of input values to one value. This case is proven by noticing that through the range of operator (..), we one represent any value or relation on values of continuous variables. QED

In concept learning, D_0, D_1, …., D_n are domains of input variables x_1,…., x_n, and D_o represents the set of concepts, or values of the out variable y_0. In general, we can have a number of output variables y_1,…., y_n , and $D^1 x \ D^2 \ x \ D^3 \ x \ ….x \ D^m$ are their domains, respectively.
QED

7. Flexible Evaluation of Attributional Calculus Expressions

In the above, attributional expressions were treated as binary logic sentences that evaluate to either true or false. Such interpretation of attributional calculus sentences is called *crisp*. In practical applications it is often unreasonable to use rules in such a strict manner. In such situations, attributional calculus expressions are evaluated flexibly, by computing a degree to which an expression is satisfied by specific facts.

There are many possible flexible evaluation schemes. A popular schema is to treat an attributional expression as a fuzzy expression, and use fuzzy logic interpretation of logic connectors. In AQ19, different interpretation schemas have been implemented (Michalski and Kaufman, 2000).

7 Summary

Attributional calculus combines aspects of propositional logic (by applying prositional operators to selectors), predicate logic (through the count attribute), and multiple-valued logic (though flexible evaluation).

Important features of attributional calculus are that it facilitates creation of descriptions that:

1. are easy to understand and interpret

2. can be simply and directly translated to natural language

3. are constitute compact representations of discrete functions

4. are gradually generalizable through internal logic operators and structured attributes

5. can be easily and efficiently implemented in a computer program.

The above features make Attributional Calculus particularly attractive for machine learning and pattern discovery.

Acknowledgments

Author thanks Ken Kaufman for valuable comments that helped to improve an earlier version of the paper.

This research was conducted in the Machine Learning and Inference Laboratory of George Mason University. The Laboratory's research activities are supported in part by the National Science Foundation Grants No. IIS 9906858 and IIS 0097476, and in part by the UMBC/LUCITE #32 grant.

References

Michalski, R.S., "A Variable-Valued Logic System as Applied to Picture Description and Recognition," *Chapter in the book, Graphic Languages*, F. Nake and A. Rosenfeld (Editors), North-Holland Publishing Co., 1972.

Michalski, R.S., "Variable-Valued Logic: System VL1, "*Proceedings of the 1974 International Symposium on Multiple-Valued Logic,"* pp.323-346, West Virginia University, Morgantown, West Virginia, May 29-31, 1974.

Michalski, R.S., "Variable-Valued Logic and Its Applications to Pattern Recognition and Machine Learning," Chapter in the monograph, Computer Science and Multiple-Valued Logic Theory and Applications, D. C. Rine (Editor), North-Holland Publishing Co., pp. 506-534, 1975.

Michalski, R.S. and McCormick, B.H., "Interval Generalization of Switching Theory," *Proceedings of the Third Annual Houston Conference on Computer and System Science*, Houston, Texas, April 26-27, 1971.

Michalski, R.S. and McCormick, B.H., "Interval Generalization of Switching Theory," (An extended version of above paper), *Report No. 442*, Department of Computer Science, University of Illinois, Urbana, May 3, 1971.

Michalski, R.S., "Pattern Recognition as Rule-Guided Inductive Inference," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-2, No. 4, pp. 349-361, July 1980.

Michalski, R.S. and Kaufman, K., "Building Knowledge Scouts Using KGL Meta-language, " *Fundamenta Informaticae*, 40, pp.433-447, 2000.

Wnek, J. and Michalski, R.S., "Experimental Comparison of Symbolic and Subsymbolic Learning," *HEURISTICS, The Journal of Knowledge Engineering*, Special Issue on Knowledge Acquisition and Machine Learning, Vol. 5, No. 4, pp. 1-21, 1992.