4. Knowledge processing

This section highlights how usual inductive learning algorithms can be stretched to complex data processing by using domain knowledge to generate accurate and meaningful decision trees (from pre-classified examples).

4.1 Tree-based classification using domain knowledge

Starting from the well known decision tree builder algorithm C4.5 [17] which works on discrete and continuous attributes, IKBS extends some functionality of this algorithm for dealing with:
1. Structured objects
2. Taxonomic attribute-values
3. Multi-valued attributes:

Let $E=\{\omega_1, ..., \omega_n\}$ be a set of observed examples, $M=\{N, Y\}$, a set of observable components and attributes defined in the descriptive model with $N=\{n_1,..., n_m\}$ a set of structured components and $A = \{A_1, ..., A_p\}$ a set of attributes depending on N.

Let dom(A) be the definition domain (range) of A.

Structured objects
The algorithm for building decision trees from structured objects is the following:

```
BuildDecisionTree (E, M)
   Y = SelectClassifier (root(M))
   BuildTree(E, Y)
end BuildDecisionTree

BuildTree(E, Y)
   if stop Criterion(E, Y) then BuildLeaf(E)
   else
     A = BestTest(E, Y) // A = y(n)
     di = BuildNode(A)
     Y = FilterClassifier(A) // depending on type of A
     partitioning(di) = R(E)
          // R(E) : ∀ ω ∈ E, Q(v1, A(ω ))=1⇔ ω ∈ Ei
     for each Ei ∈ partitioning(di)
          BuildBranch(vi)
          if (A = exist(n)) ∧ (vi = "Present"))
            Y = SelectClassifier (n)
          end if
          BuildTree (Ei, Y)
     end for each
   end if
end BuildTree
```

```
SelectClassifier (r)
   Y′ = ∅
   if (possiblyAbsent(r) = "yes") then
     Y′ = Y′ ∪ {exist(r)}
   else
     Y′ = Y′ ∪ Att(r)
     for each n_f ∈ depend(r)
          SelectClassifier (n_f)
     end for each
   end if
   return Y′
end SelectClassifier
```

The original aspect of the algorithm is the classifier's selection function. The tree of the descriptive model is followed from root to leaves, component by component in depth search first. If one of it *can* be absent (e.g. calices on verrucæ of Fig. 1), an "*exist component*" test is dynamically generated and placed in the eligible classifiers' list with values "*Present*" or "*Absent*". The sub-tree of this object is not yet visited in order to avoid inapplicable sub-objects and attributes as other classifiers. On the other hand, if components are always present (e.g. septa), dependent attributes are placed in this list.

In the identification process (see further), if the test *exist* of a component is chosen as the "best" one and the user answers that it is really present, then the classifier's selection function is recursively called on the sub-tree of the descriptive model.

Taxonomic attribute-values

For attributes which values are structured by relations of hierarchical type (classified values), an extension of the discrete classifier partitioning process is proposed.

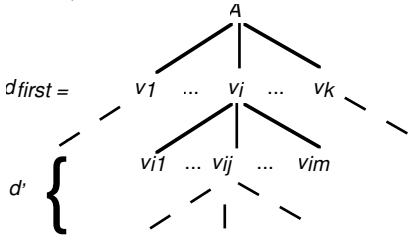

Fig. 4. Classified values of attribute A

The method consists, when such a classifier is selected, in creating a set of partitions corresponding to the first level of the hierarchy (noted d_{first}= $\{v_1,\ldots, v_i,\ldots, v_k\}$ with k elements). Each case is assigned to the partition that generalizes its value. Let A be a taxonomic attribute with the domain $d = \{v_1,\ldots,v_i,\ldots,v_n\}$ of n modalities and $d' = \{v_{i1},\ldots,v_{ij},\ldots,v_{im}\}\subset d$ is a subtree of m submodalities of v_i [Fig. 4.]:

Let Q be a Boolean application (called question) which determines if the modality v_i generalizes a value v_{ij}. Q is defined by:

$Q(v_i, v_{ij}) = 1$, if $v_{ij} \in d \cup \{v_i\}$ else $Q(v_i, v_{ij}) = 0$

Then, we can generate k partitions from d_{first}:

$E_{A1}=\{\omega \in E_A / Q(v_1, A(\omega))=1\},\ldots, E_{Ak}=\{\omega \in E_A / Q(v_k,A(\omega))=1\}$

In the next step, we create temporarily k attributes $\{A_1,\ldots, A_j,\ldots, A_k\}$ in each partition $E_{A1},\ldots,E_{Ak}$ with a set of modalities defined by the subvalues of $\{v_1,\ldots, v_i,\ldots,v_k\}$. These ones can be picked by the test function (information gain, gain ratio) and the method is recursively reapplied.

Multi-valued attributes

When modeling the descriptive model, a discrete attribute (nominal or taxonomic) can be defined as multi-valued. It can express doubt (disjunction of imprecision) or the simultaneous presence of states (conjunction of variation) like in the following expression:

$v = (v_{11}\ \&\ldots v_{1i}\ldots\&\ v_{1m}) \mid \ldots \mid (v_{j1}\ \&\ldots v_{ji}\ldots\&\ v_{jn})\ldots\mid (v_{k1}\ \&\ldots v_{ki}\ldots\&\ v_{kp})$

where $cf_j = (v_{j1}\ \&\ldots v_{ji}\ldots\&\ v_{jn})$

Depending on the semantic associated with a conjunctive form of a case (*cf*), IKBS can apply three processing methods:

1. If *cf* is true information (association of co-existing facts), create *k* partitions corresponding to each conjunction of *v*, and dispatch cases with such value in each partition: *cf* is seen as a new possible value of dom(A).
2. If *cf* expresses fuzzy information (the intrinsic variability of multiple objects is an adding source of noise), treat conjunctions as disjunction.
3. Allow the user to customize the degree of similarity ∂ between two conjunctive forms.

The default method is the third one with $\partial = 1$ because it gives a good compromise between the tree size (number of nodes) and the discrimination accuracy. Indeed, the first method don't generate a deep tree, but carries a major risk of misidentification: each *cf* of the selected attribute at a node of the decision tree must match exactly the *cf* of the tested case. The third method is more flexible because it makes a fuzzy matching for dispatching cases in each partition, depending on the number of differences between the two conjunctive forms and ∂.

4.2 Identification process

Given a set of examples, IKBS dynamically extracts the most efficient criteria from the ordered list of tests after each answer of the user. The cases are selected from this reply. If the answer is unknown, the second most discriminate test is proposed to the user, and so on. This procedure is the same as in KATE [13]. Nevertheless, IKBS processes cases that are in an object oriented formalism, while KATE starts from cases represented in a data table. In the former, the "exist (objects)" tests are directly exploitable at a node of the decision tree, while in the latter, these tests are deduced from the appearance of at least one not-applicable value in the object's attribute column of the table. Our approach is semantically better because it guarantees that the inapplicability of an attribute's value depends on the absence of an object and not the contrary.

An illustration of a decision tree built with 30 training cases is shown in Fig. 5. The numbers refer to previously discussed classifiers' types: taxonomic attributes (1 and 2 highlight the use of the same classifier at two different levels of the values' hierarchy), multi-valued attributes (case *bulbosa* with size of spines "short&long" goes in three branches (3)) and structured objects (4).