Machine Learning, 41, 259–294, 2000

Bottom-Up Induction of Feature Terms

EVA ARMENGOL eva@iiia.csic.es
ENRIC PLAZA enric@iiia.csic.es
IIIA—Artificial Intelligence Research Institute, CSIC—Spanish Scientific Research Council, Campus UAB, 08193 Bellaterra, Catalonia, Spain

Editor: Lorenza Saitta

Abstract. The aim of relational learning is to develop methods for the induction of hypotheses in representation formalisms that are more expressive than attribute-value representation. Most work on relational learning has been focused on induction in subsets of first order logic like Horn clauses. In this paper we introduce the representation formalism based on feature terms and we introduce the corresponding notions of subsumption and anti-unification. Then we explain INDIE, a heuristic bottom-up learning method that induces class hypotheses, in the form of feature terms, from positive and negative examples. The biases used in INDIE while searching the hypothesis space are explained while describing INDIE's algorithms. The representational bias of INDIE can be summarised in that it makes an intensive use of sorts and sort hierarchy, and in that it does not use negation but focuses on detecting path equalities. We show the results of INDIE in some classical relational datasets showing that it's able to find hypotheses at a level comparable to the original ones. The differences between INDIE's hypotheses and those of the other systems are explained by the bias in searching the hypothesis space and on the representational bias of the hypothesis language of each system.

Keywords: Inductive Logic Programming, relational learning, concept induction, feature structures

1. Introduction

The aim of relational learning is to develop methods for the induction of descriptions in representation formalisms that are more expressive than attribute-value representation. Relational learners are capable of dealing with structured objects, i.e. objects described structurally in terms of their components and relations among components. Learned knowledge is formed by descriptions of relations (i.e. definitions of predicates). In relational learners the languages used to represent examples, background knowledge, and concept descriptions are usually subsets of first-order logic. We think that Machine Learning research can also profit from exploring other representation formalisms that support the expressive power of relations but correspond to different subsets of first order logic. There is a group of relational learners, called *Inductive Logic Programming* systems that uses knowledge represented as Horn clauses. Most work on ILP has been focused on further subsets of Horn clause logic (Muggleton & De Raedt, 1994).

ILP research has commonly focused on concept learning, where the examples are implications and the goal is to induce a hypothesis capable of correctly classifying the examples. Concept learning uses positive and negative examples to induce a discriminating description for a concept. In this paper we introduce a representation formalism based on feature terms and INDIE, a bottom-up learning method that induces class descriptions in the form of feature terms from positive and negative examples. *Feature terms* are a generalisation of first-order terms that provides a natural way to describe incomplete information. Incomplete information arises from the so-called problem of “unknown values” in Machine Learning and, especially in attribute-value representation, the problem of irrelevant attributes. While Horn representation is based on the notion of deduction and has to define from it the notion of subsumption, feature terms representation is based around the foundational notion of subsumption and, moreover, on the notion of sort and the intensive use of sort hierarchies. The purpose of this paper is to show how to achieve relational inductive learning in this setting with results that are comparable to other relational learning systems.

The structure of this paper is the following. First, feature terms are formally described and then subsumption and anti-unification operations are defined. Subsumption defines a partial ordering between feature terms that is useful in order to compare them. The anti-unification produces as result a feature term containing all that is common to the anti-unified feature terms. Data Mining and Knowledge Discovery in Databases aim to find the properties or regularities that they present instead of discriminating descriptions (since only positive examples are considered). In particular, the anti-unification operation can be used to detect regularities between a set of feature terms. The algorithm of anti-unification is explained in detail in Section 3. In Section 4 we introduce a general view of INDIE, an inductive method for feature terms based on the subsumption and anti-unification operations. INDIE solves the discrimination task, that is to say, given a training set of positive and negative examples, INDIE finds a description satisfied (subsumed) by all positive examples and no negative example. In Section 5 the INDIE algorithm is explained in detail. In order to evaluate INDIE, we show that it is able to find correct hypotheses for several relational datasets originally proposed by other authors to evaluate their relational learning systems (Section 6). We also show the hypotheses found by the original systems and discuss the similarities and differences from the ones obtained by INDIE. Finally, we present two applications of INDIE: the identification of marine sponges in Section 7 and the assessment of risk in diabetic patients in Section 8. Our goal in using both domains is to show that INDIE is able to find hypotheses in the context of examples with partial information.

2. Feature terms

Feature terms (also called feature structures or ψ-terms) are a generalisation of first-order terms that have been introduced in theoretical computer science in order to formalise object-oriented capabilities of declarative languages (Aït-Kaci & Podelski, 1993; Carpenter, 1992). Feature term formalisms have a family resemblance with, but are different from, unification grammars and description logics (KL-One-like languages). The difference between feature terms and first order terms is the following: a first order term, e.g. $f(x,\ g(x, y),\ z)$, can be formally described as a tree and a fixed tree traversal order. In other words, parameters are identified by position. The intuition behind a feature term is that it can be described as a labelled graph, i.e. parameters are identified by name (regardless of their order or position).

Definition (Feature Terms). Given a signature $\Sigma = \langle S,\ F, \leq\rangle$ (where S is a set of sort symbols that includes $\bot$ and $\top$; F is a set of feature symbols; and $\leq$ is a decidable partial order on S such that $\perp$ is the least element and $\top$ is the greatest element) and a set ϑ of variables, we define *feature terms* as an expression of the form:

$$\psi ::= X : s\,[\,f_1 \doteq \Psi_1, \ldots,\ f_n \doteq \Psi_n] \tag{1}$$

where X is a variable in ϑ, s is a sort in S, $f_1 \ldots f_n$ are features in F, $n \geq 0$, and each Ψ_i is a set of feature terms and variables. We also identify a feature term with the singleton set of that feature term. Note that when $n = 0$ we are defining a variable without features. The set of variables occurring in ψ is noted as ϑ_ψ.

Sorts have an informational order relation ($\leq$) among them, where $\psi \leq \psi'$ means that ψ has less information than ψ'—or equivalently that ψ is more general than ψ'. Note that the informational ordering ($\leq$) is the opposite of the one usually used in ML, the "more general than" relation, while here ($\leq$) is the "less specific than" relation. The minimal element ($\perp$) is called *any* and it represents the minimum information. When a feature has "unknown value" it is represented as having the value *any*. All other sorts are more specific that *any*.

Definition (Root of a feature term). We call the variable X in definition (1) the *root* of ψ (noted Root (ψ)). Moreover, we say that ψ is *sorted* by the sort s of the root (noted Sort(ψ)) and that it has features $f_1, \ldots, f_n$.

A particular example of feature term is shown in figure 1 where X is the root of ψ_1 and variables X, Y, Z, T and P are of sort *person*.

Definition (Path). A *path* π is a sequence of features: $\pi \in F^*$.

For instance, in the feature term ψ_1 of figure 1, the path to obtain the last name of the father of person X is `X@father.last-name`—where concatenation is denoted by the dot operation and the first element left of symbol @ is the variable where the path starts.

Definition (Path equality). We say that two paths π_1 and π_2 are *equal* if both paths point to the same value.

$$\psi_1 = X : \text{person}\begin{bmatrix} \text{last}-\text{name} \doteq W : \text{family}-\text{name} \\ \text{son} \doteq \begin{array}{r} Y : \text{person}\begin{bmatrix}\text{wife} \doteq Z : \text{person}\\ \text{father}\doteq X\\ \text{brother}\doteq T\end{bmatrix} \\ T : \text{person}\begin{bmatrix}\text{father}\doteq X\\ \text{brother}\doteq Y\end{bmatrix}\end{array} \\ \text{father} \doteq P : \text{person}[\text{last}-\text{name} \doteq W] \end{bmatrix}$$

Figure 1. A feature term ψ_1 representing a *person*. This person has three features: `last-name`, `father` and `son`. The feature `son` has as value a set of two feature terms, Y and T.

Path equality is equivalent to variable symbol equality. For instance, if we look at variable symbol equality in the feature term ψ_1 in figure 1, we can see that there are two path equalities:

$$\begin{array}{rcll} \texttt{X@last-name} & = & \texttt{X@father.last-name} & \text{whose value is } W \\ \texttt{Y@father} & = & \texttt{T@father} & \text{whose value is } X \end{array}$$

Feature terms provide a way to construct terms embodying partial information about an entity. For instance, the feature term ψ_1 in figure 1 is a partial description of a person. The meaning of the feature term ψ_1 is those individuals that satisfy that partial description; i.e. ψ_1 denotes the subset of individuals such that:

1) they have a `last-name`
2) they have two sons that are of sort *person* such that
   - one son has a wife
   - both sons are brothers of each other
   - the father of both sons is the person in the root of the feature term.
3) their father is a person whose `last-name` is the same as that in 1).

2.1. Feature term subsumption

The semantic interpretation of feature terms brings an ordering relation among feature terms. We call this ordering relation *subsumption*. The intuitive meaning of subsumption is that of an *informational ordering* among partial descriptions constructed on top of the sort partial order relation ($\leq$).

Definition (Subsumption). Given two feature terms ψ and ψ', we say that ψ *subsumes* ψ', noted as $\psi \sqsubseteq \psi'$, if there is a total mapping function $\upsilon : \vartheta_{\psi} \rightarrow \vartheta_{\psi'}$ such that:

1. $\upsilon(\text{Root}(\psi)) = \text{Root}(\psi')$
   and $\forall x \in \vartheta_{\psi}$
2. $\text{Sort}(x) \leq \text{Sort}(\upsilon(x))$
3. for every $f_i \in F$ such that $x.f_i \doteq \Psi_i$ is defined, then $\upsilon(x).f_i \doteq \Psi'_i$ is also defined, and
   (a) $\forall \psi_k \in \Psi_i$, either $\exists \psi'_k \in \Psi'_i$ such that $\upsilon(\text{Root}(\psi_k)) = \text{Root}(\psi'_k)$ or $\exists x' \in \Psi'_i$ such that $\upsilon(\text{Root}(\psi_k)) = x'$
   (b) $\forall x \in \Psi_i$ either $\exists \psi'_k \in \Psi'_i$ such that $\upsilon(x) = \text{Root}(\psi'_k)$ or $\exists x' \in \Psi'_i$ such that $\upsilon(x) = x'$
   (c) $\forall \psi_k, \psi'_k \in \Psi_i (\psi_k \neq \psi'_k \Rightarrow \upsilon(\text{Root}(\psi_k)) \neq \upsilon(\text{Root}(\psi'_k)))$
   (d) $\forall x, \psi'_k \in \Psi_i (\upsilon(x) \neq \upsilon(\text{Root}(\psi'_k)))$
   (e) $\forall x, y \in \Psi_i (x \neq y \Rightarrow \upsilon(x) \neq \upsilon(y))$

As a corollary, it is worth remarking that path equality is preserved by subsumption, i.e. when $\psi \sqsubseteq \psi'$, all path equalities in ψ also occur in ψ'. Intuitively, a feature term ψ

subsumes another feature term $\psi'(\psi \sqsubseteq \psi')$ when all information in ψ is also contained in ψ'. For instance, consider the example of the feature term ψ_1 in figure 1 and the following one (ψ_2) denoting persons that have married sons:

$$\psi_2 = X_2 : \text{person}\ [\text{son} \doteq Y_2 : \text{person}\ [\text{wife} \doteq Z_2 : \text{person}]]$$

Clearly ψ_2 subsumes $\psi_1(\psi_2 \sqsubseteq \psi_1)$. Notice that in ψ_2 the `father` feature of person Y is not explicitly given and that X has only one son. Moreover for a term ψ_3 defined as

$$\psi_3 = X_3 : \text{person} \begin{bmatrix} \text{daughter} \doteq W_3 :\ \text{person} \\ \text{son} \doteq Y_3 :\ \text{person} \begin{bmatrix} \text{father} \doteq X_3 \\ \text{wife} \doteq Z_3 :\ \text{person} \end{bmatrix} \end{bmatrix}$$

it is easy to see that ψ_3 satisfies that $\psi_2 \sqsubseteq \psi_3$ (since ψ_3 has a son with a wife) but $\psi_3 \not\sqsubseteq \psi_1$ (since ψ_1 has no daughter while that information is present on ψ_3).

The subsumption relation is the inverse of the “more general than” relation:

Definition (More general than relation). Given two hypotheses h_i and h_j, h_i is more general than h_j $(h_i \geq h_j)$ if and only if any instance that satisfies h_j also satisfies h_i but some instances satisfying h_i do not satisfy h_j. (Mitchell, 1997).

Thus, when h_i *subsumes* h_j (h_i is less than h_j in the informational order, i.e. $h_i \sqsubseteq h_j$) clearly h_i is *more general than* h_j (h_i is higher than h_j in the generalization order, i.e. $h_i \geq h_j$), and one order is the inverse of the other—or, in other words, the subsumption relation is the “less specific than” relation.

2.2. *Graph and clausal syntax*

There are several syntaxes amenable to represent feature terms. We have used up to now a record—like syntax, but graph syntax and clausal syntax can also be used. Using labelled graphs, *arcs* are labelled with feature symbols, *nodes* stand for sorted variables (where the sort symbol is the node label), and *path equality* is graphically represented by arcs arriving at the same node. For instance, the graph syntax of feature term in figure 1 is shown in figure 2.

A feature term can be also understood as a conjunct of clauses—see (Aït-Kaci & Podelski, 1993) for the precise mapping among the different representations. This clausal representation is also useful and is closer to other relational learners representation. There are two kinds of atomic clauses: sort clauses, namely $s(X)$, and feature clauses, namely $f(X, Y)$. Thus, the clausal representation of a feature term is as a conjunction of these two kind of atomic clauses. The clausal form of a term $\psi ::= X : s[f_1 \doteq \Psi_1 \ldots f_n \doteq \Psi_n]$ is built with a transformation Φ as follows: $\Phi(\psi) = s(X) \wedge f_1(X, Y_1) \wedge \Phi(\Psi_1) \wedge \ldots \wedge f_n(X, Y_n) \wedge \Phi(\Psi_n)$ where $Y_1, \ldots, Y_n$ are the roots of $\Psi_1, \ldots, \Psi_n$ respectively. When a feature value is a set Ψ_1 then the Φ transformation is applied to each element $\psi_w \in \Psi_i$, as follows: $\Phi(\psi_w) = f_1(X, W) \wedge \Phi(\psi_w)$ where W is the root of ψ_w.

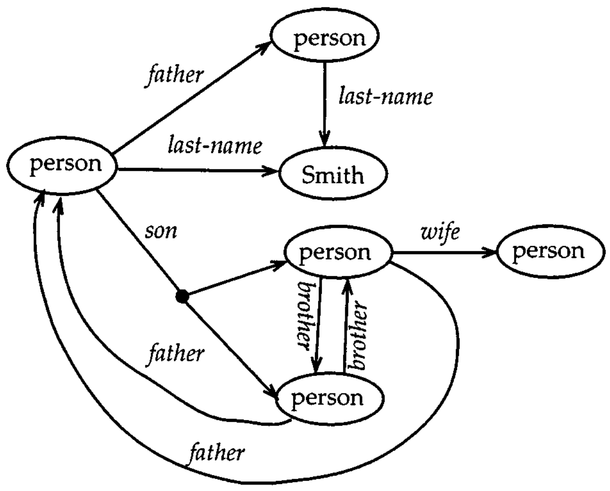


Figure 2. Representation of the feature term of figure 1 using a directed graph. Notice that the arc *son* has as value a set of two persons.

For instance, the feature term in figure 1 can be represented in clausal form as follows:

$$\begin{aligned}&\text{person}\ (X) \wedge \text{last-name}\ (X, W) \wedge \text{family-name}\ (W)\\ &\quad \wedge \text{son}\ (X, Y) \wedge \text{person}\ (Y) \wedge \text{wife}\ (Y, Z)\\ &\quad \wedge \text{person}\ (Z) \wedge \text{father}\ (Y, X) \wedge \text{brother}\ (Y, T)\\ &\quad \wedge \text{son}\ (X, T) \wedge \text{person}\ (T) \wedge \text{father}\ (T, X) \wedge \text{brother}\ (T, Y)\\ &\quad \wedge \text{father}\ (X, P) \wedge \text{person}\ (P) \wedge \text{last-name}\ (P, W)\end{aligned}$$

3. Anti-unification

Many algorithms in Machine Learning use the *more general than* relation to organise the search space. Feature terms form a partial ordering by means of the subsumption relationship. By starting from subsumption it is natural to define the operations of unification and anti-unification. In particular, to introduce the anti-unification operation we need first to define *equivalence* among feature terms as follows:

Definition (Syntactic Variants). Given two feature terms ψ and ψ' we say that they are *syntactic variants* if and only if $\psi \sqsubseteq \psi'$ and $\psi' \sqsubseteq \psi$.

Two feature terms that subsume each other are equivalent with respect to the informational order in that they both contain the same information: they are equal up to a renaming of variables. The *anti-unification* in feature terms is defined in the classical way (as the *least common subsumer* or *most specific generalisation*) over the subsumption lattice as follows:

$$\text{a)} \left\{ \begin{array}{l} \text{person1} = \text{P1} : \text{person} \left[\begin{array}{l} \text{name} \doteq N_1 : \text{name} \left[\begin{array}{l} \text{first} \doteq \text{John} \\ \text{last} \doteq \text{Smith} \end{array} \right] \\ \\ \text{lives} - \text{at} \doteq A_1 : \text{address} \left[\text{city} \doteq \text{NYCity} \right] \\ \text{father} \doteq X_1 : \text{person} \left[\text{name} \doteq \text{Smith} \right] \end{array} \right] \\ \\ \text{person2} = \text{P2} : \text{person} \left[\begin{array}{l} \text{name} \doteq N_2 : \text{name} \left[\text{last} \doteq \text{Taylor} \right] \\ \text{wife} \doteq Y_2 : \text{person} \left[\text{name} \doteq M_2 : \text{name} \left[\text{first} \doteq \text{Mary} \right] \right] \\ \text{father} \doteq X_2 : \text{person} \left[\text{name} \doteq \text{Taylor} \right] \end{array} \right] \end{array} \right.$$

$$\text{b)} \qquad \text{P3} : \text{person} \left[\begin{array}{l} \text{name} \doteq N_3 : \text{name} \left[\text{last} \doteq \text{family} - \text{name} \right] \\ \text{father} \doteq X_3 : \text{person} \left[\text{name} \doteq \text{family} - \text{name} \right] \end{array} \right]$$

Figure 3. An anti-unification example where (a) shows two feature terms representing *person1* and *person2*, and (b) shows the feature term *P3* obtained by their anti-unification.

Definition (Anti-unification). The *anti-unification* of two feature terms ($\psi \sqcap \psi'$) is a greatest lower bound (glb) with respect to the subsumption ($\sqsubseteq$) ordering.

Intuitively, the anti-unification of two feature terms gives what is common to both (yielding the notion of generalisation) and all that is common to both (the most specific generalisation).

Example 1. Let *person1* and *person2* be the objects represented as the feature terms in figure 3(a). The anti-unification of both is the feature term P3 shown in figure 3(b). The sort of P3 is *person* and its features are those features common to *person1* and *person2* (namely `name` and `father`). In P3, the feature `last` of the `name` feature term has as value the sort *family-name* that is the greatest lower bound in the sort hierarchy according to the $\leq$ sort relation, i.e. the most specific sort common to both *Taylor* and *Smith* values. Features `wife` and `lives-at` only appear in one of the terms in figure 3(a), therefore they do not appear in the anti-unification feature term P3.

Formally, when a feature does not appear in a feature term, it is equivalent to consider that this feature has value *any*, the minimal sort according to the $\leq$ sort relation (see the definition of feature terms). In such situation, the anti-unification of *any* with another value produces as result *any*, thus the feature will not "appear" in the anti-unified feature term.

Path equality `person1@person.name.last=person1@person.father.name.last` of feature term *person1* also occurs in *person2*. For this reason, the path equality is preserved in the anti-unification feature term *P3*. In general, a feature term obtained by the anti-unification of a set of feature terms will contain a path equality only if all the anti-unified feature terms contain that path equality.

```
D : set of feature terms.
feat(d) returns the set of features of the root of the feature term d
Initially: *paths* := $\varnothing$; $E = e_1 \ldots e_n$;
function AU (E)
1  Let d be a term with $Sort(d) = Sort(e_1) \sqcap Sort(e_2) \sqcap \ldots \sqcap Sort(e_n)$
        and $feat(d) = \varnothing$
2   Add pair (E, d) to *paths* and D:={d}
3   $A := \{a_i \mid a_i \in feat(e_i), \forall e_i \in E\}$
4   for each $a_i \in A$ do
5     $W_i = (v_{i1}, \ldots, v_{in})$ where $v_{ij} = e_j.a_i$ $\forall e_j \in E$   ;$v_{ij}$ can be a set
6     if $\forall v_{ij}, v_{ik} \in W_i$, $v_{ij} = v_{ik}$
7       then add-feature-to-all(D, $a_i$, $v_{ij}$)
8       else
9         if there is a $p = (W_j, d_j) \in$ *paths* such that $W_i = W_j$
10        then add-feature-to-all(D, $a_i$, $d_j$)
11        else if $\exists v_{ij} \in W_i$ Card $(v_{ij}) > 1$
12               then $D_i$ := Antiunify-sets ($W_i$)
13               else $D_i$ := AU ($W_i$)
14        endif endif endif
15   if Card ($D_i$) = 1
16   then add-feature-to-all(D, $a_i$, $D_i$)
17   else for each $d_k \in D$ do
18               m := Card ($D_i$)
19               M := mcopy($d_k$, m)        ; produces m copies of $d_k$
20               for j:= 1 to m do
21                 add-feature($m_j$, $a_i$, $d_j$) $\forall m_j \in M, \forall d_j \in D'_i$ end-for
22               D := D + M
23        end-for
     end-if end-if end-if end-for
   return D
end-function
```

Figure 4. The AU algorithm constructs the most specific generalisation covering a given set of positive examples. The function *Add-feature*(d, a, v) adds the feature a with value v to the feature term d. The function *Add-feature-to-all*(D, a, v) adds the feature a with value v to all the feature terms in the set D. The variable *paths* is a list of pairs (W_i, d_i) where W_i is a set of already anti-unified feature terms and d_i is their anti-unification.

3.1. Anti-unification algorithm

As we will see later, when feature terms have sets of values in some feature, anti-unification may be not unique. Figure 4 shows the anti-unification algorithm (AU) used to obtain a most specific generalisation of a set of examples.

Given a set of examples $E = \{e_1 \mathinner{..} e_n\}$ represented as feature terms, the AU algorithm builds a new feature term D following three steps:

1) the sort of the root of D is the most specific sort common to all the sorts of the roots of the examples in E,
2) the set A of features present in D is formed by those features present in all the examples in E,
3) the value of each feature a_i in A is computed by recursively applying AU to the set formed by the values that a_i takes in each $e_k \in E$.

We will introduce the AU algorithm with an example. For each feature $a_i \in A$ common to all $e_k \in E$, let us consider the set $W_i = (v_{i1}, \ldots, v_{in})$ where $v_{ik} = e_k . a_i$, i.e. the value taken by the a_i feature in the example e_k.

The first case in the AU algorithm (line 6 in figure 4) is to check if $v_{i1} = v_{i2} = \cdots = v_{in}$, i.e. whether all the examples have the same value in the feature a_i. In this situation the anti-unification is a feature term that has exactly that value in feature a_i. A second case (line 9) considering the set $W_i = (v_{i1}, \ldots, v_{in})$ is when there is a path equality, i.e. when two (or more) features of a term have the same value. If this path equality is shared by all the terms in E, the feature term resulting from the anti-unification will have the same path equality (see Example 1). Detecting a path equality means that the same set W_i has already been antiunified by the algorithm. In the implementation, we use the **paths** variable (see figure 4) that contains all the pairs (W_j, d_j) already processed and where d_j is the feature term generated by anti-unifying the values of the set W_j. Therefore, for a given set W_i the algorithm searches in **paths** for a pair (W_j, d_j) such that $W_j = W_i$. If this pair is found it means that there is a path equality in the anti-unified term D, and the value for feature a_i has to be exactly d_j.

Finally, the third case (line 11) in the AU algorithm holds when there is no set W_j in **paths** such that $W_j = W_i$. In that situation, the AU algorithm distinguishes two cases:

1) there is at least one term $e_k \in E$ such that $e_k . a_i = V$ where the current feature a_i has a value V that is a set, or
2) none of the values of $e_k . a_i$ is a set.

The second case simply calls recursively the AU function, as shown on line 13 in figure 4.

The first case is solved using the *antiunify-sets* function (see figure 5). Let $W = (V_1, \ldots, V_n)$ be the collection of values V_k where each V_k is the value that the current feature a_i takes in an example e_k. We express this situation as $e_k . a_i = V_k$. Each V_k is a set, although some can be singleton sets. *Antiunify-sets* has to produce a set S, where each element in S is the anti-unification of one element in each $V_i \in W$. Each $V_i = (x_{i1}, \ldots, x_{iN_i})$ is a set of terms x_{ij} where $1 \leq j \leq N_i$ and $N_i = \text{Card}(V_i)$. Each V_i has different cardinality and according to the subsumption definition (Section 2), the cardinality of S has to be $\text{MinCard} = \min\{\text{Card}(V_i)\}$ where $i = 1, \ldots, n$.

Elements in S are obtained as follows. First a set C is built where each element in C is a tuple $(x_{1j_1}, \ldots, x_{nj_n})$ obtained from the Cartesian product $V_1 \times \cdots \times V_n$. In other words, each element x_{ij_k} of a tuple is the j_k-th element of the set V_i. There are Card $(V_1) \times \cdots \times$ Card (V_n) possible tuples of values from all V_i.

From the set C, the algorithm obtains the subset CS containing those tuples in C having the most specific root sort. Next, the set DS is built containing the terms d_j obtained from

```
Function ANTIUNIFY-SETS (W)
  Let n = Card (W)
  MinCard := min(Card($V_i$)), $\forall V_i \in$ W
  Choose one $V_k \in$ W such that Card($V_i$) = MinCard
  C := {($x_{1j_1} \ldots x_{nj_n}$) | $x_{ij_k} \in V_i$ and $1 \le j_k \le$ Card ($V_k$)}
  search-most-specific (C, MinCard)
end-function
```

```
Function SEARCH-MOST-SPECIFIC (C, MinCard)
CS := {$c_j \in$ C| $\forall c_k \in$ C : glb($c_j$) $\sqsubseteq$ glb($c_k$)}
      where glb(c) = sort($x_1$) $\sqcap \ldots \sqcap$ sort($x_h$) for c=($x_1$, ... $x_h$)
DS := {$d_j$ | $d_j$ = AU ($c_j$) $\forall c_j \in$ CS}
Add all ($c_j$, $d_j$) to *paths*
  Let MS = {$d_i \in$ DS | no $\exists$ $d_k \in$ DS : $d_k \sqsubseteq d_i$}
  cases
    • Card (MS) = 1 : return Obtain-one-solution ($d_i$, C, MinCard)
    • Card(MS) > 1 : return Obtain-several-solutions(MS, C, MinCard)
  end-cases
end-function
```

Figure 5. *Antiunify-sets* takes a collection W of sets of values and produces as result its anti-unification. glb(c) represents the greatest lower bound sort of a tuple c_i of sorts to be anti-unified.

```
Function OBTAIN-ONE-SOLUTION ($d_i$, C, MinCard)
  if MinCard = 1 then return $d_i$
                 else Let $c_i$ be the tuple in C associated to $d_i$
                      Solution:= {$d_i$}
                      CS := compatible-tuples ($c_i$, C)
                      Solution:= Solution $\cup$ search-most-specific (CS, MinCard-1)
  endif
end-function
```

Figure 6. Algorithm of the *Obtain-one-solution* function. This algorithm is used to obtain a set of MinCard terms compatible with a fixed term d_i.

the anti-unification of the elements (feature terms) of each tuple $c_j \in$ CS. Finally, from DS, the set of most specific terms (MS) is built. MS contains those terms in DS that are not subsumed by another term is DS. If the set MS contains only one term then the anti-unification of the set W will be unique and it will be obtained using the *obtain-one-solution* function (figure 6). Otherwise, values in W can produce several antiunifications. In this case, the anti-unification is obtained using the *obtain-several-solutions* function shown in figure 7.

Let us analyse the *obtain-one-solution* function of figure 6. The term d_i is an element of S but we still need MinCard - 1 elements more to completely build S. Once a term d_i

```
Function OBTAIN-SEVERAL-SOLUTIONS (MS, C, MinCard)
  solution := ∅
  for each d_k ∈ MS do
        solution := solution ∪ obtain-one-solution (d_k, C, MinCard)
  end-for
  return solution
end-function
```

Figure 7. Algorithm of the *obtain-several-solutions* function. This algorithm is used to obtain all the solutions when anti-unification is not unique.

obtained from a tuple c_i has been included in S, all the tuples in C incompatible with c_i may be eliminated. We say that two tuples are *compatible* if their intersection is empty, i.e. they do not have any common element. Otherwise the tuples are *incompatible*. A tuple c_j is incompatible with c_i if there is some value in tuple c_j that is also in c_i, and thus, it has already used in the anti-unification that obtained d_i. Let us illustrate this definition with an example. Let us suppose that objects X_1 and X_2 have the following definitions:

$$X_1 : \text{person}\left[\text{children} \doteq \begin{matrix}\text{Mary}\\ \text{John}\end{matrix}\right] \qquad X_2 : \text{person}\left[\text{children} \doteq \begin{matrix}\text{Peter}\\ \text{Jennifer}\end{matrix}\right]$$

The sets of values of the children feature in both feature terms can be combined in the following tuples: $c1 = (Mary, Peter)$, $c2 = (Mary, Jennifer)$, $c3 = (John, Peter)$ and $c4 = (John, Jennifer)$. Tuple $c1$ is incompatible with $c2$ since both share the value *Mary*, and $c1$ is also incompatible with $c3$ since both share the value *Peter*. Pairs of compatible tuples are ($c1$, $c4$) and ($c2$, $c3$).

When *search-most-specific* function of figure 5 has Card(MS) $>$ 1 there are several terms in MS that are maximally specific. Consequently, *search-most-specific* will provide a different anti-unification for each subset of term $d_k \in$ MS compatible with each d_i. *Search-most-specific* uses *obtain-several-solutions* function to obtain all possible solutions. In turn, the *obtain-several-solutions* function (figure 7) uses *obtain-one-solution* to obtain, for each maximally specific term $d_i \in$ MS, MinCard terms compatible with d_i. In this case *search-most-specific* will return several feature terms as the anti-unification of sets in W.

When the anti-unification of a set W is not unique, the AU algorithm (lines 19 to 23 in figure 4) has to create M copies of the current anti-unification term D. Each copy will contain the current feature a_i with a different value according to the result of *obtain-several-solutions*.

We have now finished the detailed presentation of the anti-unification algorithm. Since INDIE is a bottom up learner that is based on the anti-unification operation and the subsumption lattice this presentation was necessarily a detailed one. However, as a summary, it's worth noting that, except for the management of set-valued features, the AU algorithm is straightforward: AU is a recursive function that for each feature computes the sort of the next node as the least common sort and keeps track of the path equalities already visited.

4. General view of INDIE

In this section we describe INDIE, an inductive learning method for the discrimination task, defined as follows:

Given: a set E containing positive E^+ and negative E^- examples, a notion of subsumption, and background knowledge
Find a term D such that $\forall e \in E^+ : D \sqsubseteq e$ and $\forall e' \in E^- : D \not\sqsubseteq e'$

In other words, a discriminating term D subsumes all positive examples and does not subsume any negative example. Let us assume that training examples $E = E^+ \cup E^-$ are classified in M solution classes $C_1, \ldots, C_M$. The goal of INDIE in the discrimination task is to build a hypothesis D for a solution class C_k such that D subsumes all the positive examples and does not subsume any negative example. Positive examples of the solution class C_k are those training examples classified as belonging to C_k and negative examples of C_k are those belonging to solution classes different than C_k. We assume that the training examples are correctly and uniquely classified.

Figure 8 shows the knowledge modelling of INDIE in the discrimination task. INDIE is decomposed in two subtasks: `induction` and `simplification`. The `induction` task uses knowledge from positive and negative examples to obtain a hypothesis for a solution class. The `simplification` task generalises as much as possible the hypothesis obtained by the `induction` task using a heuristic method for feature elimination (explained in Section 5.3). The `simplification` task is optional. The `induction` task is solved using the `bottom-up-induction` method. This is the heuristic bottom-up method that builds a hypothesis that subsumes the positive examples and does not subsume the negative examples. This method is made up of two subtasks: `generalisation` and `specialisation`. The `generalisation` task uses the `anti-unification` method explained in the previous section to obtain a most specific generalisation D subsuming all the positive examples.

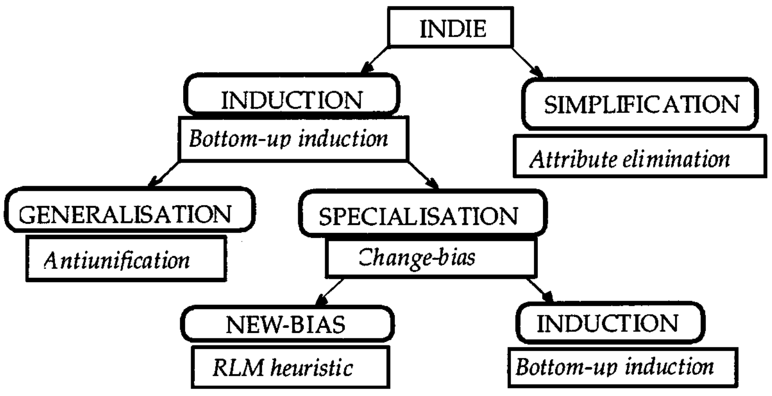


Figure 8. Decomposition of INDIE method for the `discrimination` task.

If the hypothesis D obtained by `generalisation` task also subsumes some negative examples, the `specialisation` task is used to specialise D. Specialisation (see Section 5.1) is achieved by means of the `change-bias` method that replaces D with a disjunction of terms. This method decomposes in two subtasks: `new-bias` and `induction`. The `new-bias` task decides how many disjuncts are necessary using the López de Mántaras distance (López de Mántaras, 1991). For this purpose, the `RLM-heuristic` method (Section 5.2) partitions the set of positive examples in N sets. Finally, the `induction` task is newly applied to each set of the partition in order to find a suitable discriminating hypothesis; the result will therefore be a hypothesis formed by a disjunction of terms. This process is repeated until a hypothesis that does not subsume negative examples is found.

5. Description of the INDIE algorithm

Given a set of training examples $E = \{e_1, \ldots, e_n\}$ and a set of solution classes $C = \{C_1, \ldots, C_M\}$, the goal of INDIE is to obtain a discriminating hypothesis D for each solution class C_k. Each example e_i is a feature term having a subset of features $A_i = \{A_{i1}, \ldots, A_{ip} \mid A_{ij} \in F\}$. A hypothesis $D_k = \{d_k^j\}$ represents a disjunction of feature terms describing the current solution class C_k since it subsumes all the positive examples of C_k. Each d_k^j subsumes a subset of positive examples of C_k and does not subsume any negative example. In a discrimination task, negative examples of a solution class C_k are all those training examples that do not belong to C_k.

Given a set of positive examples E^+ for a solution class C_k the `bottom-up-induction` algorithm (figure 9) obtains, using the anti-unification operation, a most specific generalisation D_k subsuming all the examples in E^+. There are two possible cases: (1) the anti-unification of E^+ is a unique feature term, or (2) the anti-unification of E^+ is a collection of feature terms. This second case occurs when antiunification is not unique due to the presence of set-valued features among the positive examples. To solve the first case the `specialisation` function (figure 10) is used. The second case is solved using the

```
D = ∅
Function BOTTOM-UP-INDUCTION  (E⁺, E⁻, D)
  Dk = AU(E⁺) ; most specific generalisation
  case
    • Card (Dk) = 1 : return specialisation (E⁺, E⁻, Dk, D)
    • Card (Dk) > 1 : solutions := ∅
                      for each dʲk ∈ Dk do
                        solutions := solution ∪ Specialisation (E⁺, E⁻, dʲk, D)
                      end-for
                      return solutions
  end-case
end-function
```

Figure 9. The Bottom-up-Induction function obtains a set of feature terms that do not subsume negative examples for the current class.

```
Function SPECIALISATION ($E^+$, $E^-$, $d_k$, D)
  if there is some $e \in E^-$ such that $d_k \sqsubseteq e$
    then $A_L$ = {$a_i$ | features in $d_k$ chosen according to a bias}
         $P_{dis}$ = RLM-Heuristic ($A_L$, $E^+$, $E^-$)
         for each set $S_i \in P_{dis}$ do    $D_i$ = BOTTOM-UP-INDUCTION ($S_i$, $E^-$, $\emptyset$);
                                              Add $D_i$ to D end-do end-for
    else Add $d_k$ to D end-if
  Eliminate any $d_j \in$ D such that $d_j \sqsubseteq d'$ for any $d' \in$ D
  return D
end-function
```

Figure 10. Specialisation function obtains a new disjunctive hypothesis that does not subsume negative examples.

`specialisation` function for each feature term in D_k, and thus performing an exhaustive search. Let us now analyse the `specialisation` function. If the current hypothesis d_k does not subsume any negative example then d_k is a correct hypothesis and INDIE's goal has been achieved. If d_k subsumes some negative example it needs to be specialised. However, d_k is already a most specific generalisation of the positive examples; consequently, the only way to specialise d_k is to transform it into a disjunctive hypothesis. INDIE has to partition E^+ into a collection of sets of examples and then find a term for each of these sets that does not subsume any element of E^-. In order to do so, INDIE selects a feature a_d and partitions the examples of E^+ according to the sort of the value of a_d in each example.

Thus, the main goal of the specialisation process is to determine the best feature to be used to specialise the current hypothesis. This issue is addressed by an heuristic approach: the López de Mántaras distance (López de Mántaras, 1991). The López de Mántaras (RLM) distance calculates the distance between a partition over the examples defined by an attribute and the correct partition. The correct partition in INDIE is formed by two sets, one containing the positive examples and another with the negative examples. Each feature $a_i \in A_L$ induces one or more partitions over the set of training examples, according to the sorts that the feature a_i takes on the examples (see Section 5.2 for a more detailed explanation). INDIE uses the `RLM-Heuristic` function to select the most discriminating feature a_d. Such feature induces a partition P_{dis} of the training examples such that the RLM distance between P_{dis} and the correct partition is minimal. Next, INDIE uses this most discriminating feature $a_d \in A_L$ to induce a partition P_{dis} that has m sets, where m is the number of different sorts that a_d takes in E^+. This partition is returned to the `specialisation` function (figure 10) where the `bottom-up-induction` function is recursively applied to each set of the partition P_{dis}. At this level, INDIE will generate a disjunct of (at most) m terms (one for each partition set) as new hypothesis. In general, one of these sets can be further partitioned into subsets and thus the number of disjuncts may be greater than m. This process is repeated until the hypothesis does not subsume any negative example. The final disjunctive hypothesis D is composed of several disjuncts d_k some of which may be redundant, i.e. two terms d_k and d_j in D can satisfy that $d_k \sqsubseteq d_j$. For this reason the last instruction of the `specialisation` function is the elimination of such redundant terms. In particular, when $d_k \sqsubseteq d_j$ the term d_j is eliminated.

$$\text{Francesca} : \text{female}\left[\begin{array}{l} \text{husband} = \text{Piero}\left[\begin{array}{l}\text{wife} = \text{Francesca}\\ \text{hair} = \text{blond}\end{array}\right]\\ \text{son} = \text{Marco}\left[\begin{array}{l}\text{mother} = \text{Francesca}\\ \text{father} = \text{Piero}\\ \text{wife} = \text{Mary}\left[\text{brother} = \text{Albert}\right]\\ \text{hair} = \text{brown}\end{array}\right]\\ \text{hair} = \text{brown}\end{array}\right]$$

Figure 11. A feature term describing *Francesca*.

There is a special case in which the partition induced by feature a_d over E^+ has only one set; this is possible since RLM works on partitions over E, not only E^+, and there may be a partition of E where all positive examples are in just one set. INDIE deals with this special case by simply rejecting the current $a_d \in A_L$ and using the next more discriminating feature in A_L to proceed.

In the next sections, we explain the bias used to select the set of features candidates to partition the set of examples (Section 5.1) and how the most discriminating feature is selected (Section 5.2).

5.1. INDIE's specialisation bias

In order to determine the most discriminating feature INDIE uses two main biases on the specialisation process. First, of all possible features in F, INDIE will consider only those features present in the current hypothesis D_k—i.e. the result of the anti-unification. As a consequence, any feature that is not present in the current hypothesis will not be considered by the RLM heuristic of Section 5.2.

The second bias is a depth threshold that determines the maximum depth at which a feature can appear in a term in the hypothesis to be considered eligible by the RLM heuristic. For this purpose we need first to define the notion of feature depth.

Definition (Depth of a feature). The *depth of a feature* f in a feature term Ψ with root X is the number of features that compose the path from the root X to f, including f, with no repeated nodes.

Notice that this definition deals with feature depth and not with node depth. We'll explain this notion using as example the term shown in figure 11. If we consider the path `Francesca@son` we see that feature `son` has depth 1, while considering the path `Francesca@husband.wife` we see that feature `wife` has depth 2. Notice that a particular node can be reached by more than one path when there is a path equality. For instance, in figure 11, *Francesca* occurs in three places: at the root (the empty path), as value of path `Francesca@husband.wife` and as value of path `Francesca@son.mother`. For this reason we do not define node's depth but feature depth: depth is a property of paths and not of nodes.

Lastly, the proviso in feature depth definition that the path should not contain repeated nodes is included to avoid circularities. Let us consider the following path: $\pi_1 =$ `Francesca@husband.wife.husband.wife`. The value of π_1 is *Francesca*. However, path π_1 has another occurrence of the node Francesca as the value of subpath $\pi_2 =$ `Francesca@husband.wife`. According to the feature depth definition, the depth of `wife` is determined only by path π_2 where *Francesca* occurs only once.

Let us now transform feature depth into a bias. INDIE considers feature candidates for specialisation only those features in the term produced by the AU operation. This second bias restricts this set of candidates to those that are leaf features. INDIE has a specific parameter called maximum feature depth, Δ, to control this bias.

Definition (Leaf feature). Given a particular maximum feature depth Δ, a *leaf feature* is (1) a feature with depth Δ or (2) a feature with lesser depth that has as value a node without features.

Thus, the set of feature leaves L_Δ of the term in figure 11 with $\Delta = 1$ is $L_1 =$ {`Francesca@husband`, `Francesca@son`, `Francesca@hair`}, with $\Delta = 2$ it's the set $L_2 =$ {`Piero@wife`, `Piero@hair`, `Marco@mother`, `Marco@father`, `Marco@wife`, `Marco@hair`}, and with $\Delta = 3$ it's the set $L_3 =$ {`Piero@wife.husband`, `Piero@wife.son`, `Piero@wife.hair`, `Marco@mother.husband`, `Marco@mother.son`, `Marco@mother.hair`, `Marco@father.wife`, `Marco@father.hair`, `Mary@brother`}. For a given depth and a current hypothesis D_k these biases define A_L, the set of candidate features to specialise D_k that will be used by the RLM heuristic of the next subsection.

5.2. The RLM heuristic

In this section we explain the heuristic to select the most discriminating feature for specialisation. Figure 12 shows the algorithm based on evaluating the distance between the correct partition and the partitions induced by the features in the set A_L obtained as explained in previous section.

Let D be the hypothesis obtained from the anti-unification of the set of positive examples E^+ that also subsumes some negative example. Now INDIE has to transform D into a disjunctive hypothesis. For this purpose, INDIE has to define a partition of the set E^+ in subsets $E_1, \ldots, E_s$ to which the `Bottom-up-induction` algorithm will be recursively applied. The RLM heuristic decides which specific partition will be used.

Let $A_L = \{a_1 \,.\,.\, a_n\}$ be the set of features that can be used to induce a partition, as determined by the bias explained in Section 5.1. INDIE selects a most discriminating feature $a_d \in A_L$ using the López de Mántaras (RLM) distance (López de Mántaras, 1991). The partition is induced over the current training set, namely $E = E^+ \cup E^-$, where E^+ is the currently considered subset of positive examples and E^- is the set of all negative examples. Since each feature $a_i \in A_L$ induces a partition P_i over the training set E according to the sorts that a_i can take in E, the RLM heuristic is used to determine the feature that induces a partition that is closer to the correct partition.

Basically, the partition P_i induced by a feature a_i is built according to the number of "different values" that a_i takes in the set E^+. Thus, examples belonging to a set of the

```
Function RLM-HEURISTIC ($A_L$, $E^+$, $E^-$)
  Dist = $\emptyset$
  while $A_L \neq \emptyset$ do
      $P_c$ = (($E^+$)($E^-$))   ;; the correct partition
      for $a_i \in A_L$ do $P_i = \{S_i \subset E \mid \forall v_i \in S_i$ and $\forall v_j \in S_j$: Sort($v_i$) $\neq$ Sort($v_j$)$\}$
                  $D_i$ = D($P_c$, $P_i$)   ;; López de Mántaras distance
                  Add $D_i$ to Dist
      end-for  end-while
  while Dist $\neq \emptyset$ and (useful-feature = false) do
      $d_{min}$ = min $\{D_i \in$ Dist$\}$
      Let $a_{min}$ and $P_{min}$ be the feature and the partition associated to $d_{min}$
      $P_d = \{S'_i \mid S'_i = S_i - E^-: S_i \in P_{min}\}$
      if $P_d$ has only one non-empty $S'_i$ then remove $d_{min}$ from Dist
                                  else  useful-feature = true endif
  end-while
  if Dist = $\emptyset$ then return $E^+$ else return $P_d$ end-if
end-function
```

Figure 12. Discriminating-partition function selects the most useful feature in a term leaf using the López de Mántaras distance.

partition P_i have the same value in the feature a_i. Since in feature terms values are sorted, INDIE can consider values "equal" or "different" depending on whether they have or not a sort in common. In other words, for each feature a_i several partitions $\{P_{ik}\}$ are generated. Each partition P_{ik} is induced according to different combinations of the sorts to which the possible values of A_i belong.

Let us suppose that a feature a_i takes in E the values v_1, v_2 and v_3 that have sorts S_1, S_2, and S_{12} according to the hierarchy of sorts of figure 13 (notice that v_4 is not present in the examples). In addition to the partition induced by v_1, v_2 and v_3, INDIE considers the following partitions:

- (S_1, v_2), i.e. the training set is partitioned in two groups, one containing examples whose value in feature a_i has sort S_1 and the other containing examples whose value in the feature a_i is v_2.
- (v_1, v_3, S_2), i.e. the training set is partitioned in three groups, one containing examples whose value in feature a_i is v_1; a second group whose value is v_3, and a third whose value in feature a_i has sort S_2.

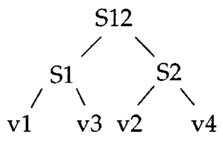


Figure 13. An example of sort hierarchy.

- (S_1, S_2), i.e. the training set is partitioned in two groups. One group contains examples whose value in feature a_i has sort S_1, and the other whose value has sort S_2.

Thus, for each $a_i \in A_L$, INDIE considers all the partitions $\{P_{ik}\}$ that are meaningful with respect to the sort hierarchy. For each partition P_{ik} INDIE computes the RLM distance of P_{ik} with respect to the correct partition. INDIE will consider the best a_i as the feature with a partition P_{ik} that has the least distance to the correct partition.

The RLM measures the distance between P_i and the correct partition as follows: Given two partitions P_A and P_B of a set X, the RLM distance between them is computed as follows:

$$\mathrm{RLM}(P_\mathrm{A}, P_\mathrm{B}) = 2 - \frac{I(P_\mathrm{A}) + I(P_\mathrm{B})}{I(P_\mathrm{B} \cap P_\mathrm{A})} \quad \text{where } I(P_\mathrm{A}) = -\sum_{i=1}^{n} p_i \log_2 p_i,$$

$$p_i = \frac{|X \cap C_i|}{|X|}$$

$$I(P_\mathrm{A} \cap P_\mathrm{B}) = -\sum_{i=1}^{n}\sum_{j=1}^{m} p_{ij} \log_2 p_{ij},$$

$$p_{ij} = \frac{|X \cap C_i \cap C_j|}{|X|}$$

where $I(P_\mathrm{A})$ measures the information contained in the partition P_A; $n(m)$ is the number of possible values of the feature inducing $P_\mathrm{A}(P_\mathrm{B})$; p_i is the probability of occurrence of class C_i in the set of examples X, i.e. the proportion of examples in X that belong to C_i; $I(P_\mathrm{A} \cap P_\mathrm{B})$ is the mutual average information of two partitions; and p_{ij} is the probability of occurrence of the intersection $C_i \cap C_j$, i.e. the proportion of examples in X that belong to C_i and to C_j.

Given two partitions, the RLM heuristic provides the following relation among features:

Definition (More discriminating feature). Let P_c be the correct partition, and P_j and P_k the partitions induced by features a_j and a_k respectively, we say that feature a_j is *more discriminating than* feature a_k iff $\mathrm{RLM}(P_\mathrm{c}, P_\mathrm{j}) < \mathrm{RLM}(P_\mathrm{c}, P_\mathrm{k})$.

In other words, when a feature a_1 is more discriminating than another feature a_2 the partition that a_1 induces over the set of examples is closer to the correct partition P_c than the partition induced by a_2. Therefore, the RLM-Heuristic returns the feature a_d inducing a partition with the least distance to the correct partition.

5.3. *Simplification post-process*

After INDIE has induced a discriminating class hypothesis D, an optional post-processing step can be used. This post-process is similar to the one used by FOIL (Quinlan, 1990). Since INDIE is a bottom-up induction method, the disjunctive hypothesis $D = \{d_k^j\}$ obtained for

```
Function ATTRIBUTE-ELIMINATION (E⁻, D)
  For each term $d^j{}_k \in D$ do
    $A_o = (a_1 \ldots a_n)$;Features in $d^j{}_k$ ordered using the RLM heuristic
    For each $a_i \in A_o$ (i = 1 to n) do
      $d_{new} := d - a_i$
      if there is no $e \in E^-$ such that $d_{new} \sqsubseteq e$ then
        $d := d_{new}$
      end-if
    end-for
end-for
eliminate-redundancies(D)
end
```

Figure 14. The `Attribute-elimination` algorithm that eliminates features from a current hypothesis obtained using the INDIE method.

a class C_k is a most specific generalisation for C_k that does not subsume examples of other classes. The hypothesis can be further generalised in so far as no negative example is subsumed. The generalisation algorithm for post-processing is shown in figure 14. For each feature term d_k^j in the hypothesis D, the algorithm *Attribute-elimination* uses the López de Mántaras distance to rank all the features belonging to d_k^j. The features are considered from the least discriminating to the most discriminating. Following this order, one step in the algorithm considers a new term generated by eliminating the least discriminating feature from term d_k^j. If the new term does not subsume negative examples then the least discriminating feature can be eliminated, and the new term substitutes the old d_k^j. All the features are explored in this order, and the final result is a term containing the features that are necessary to identify the examples of the current class C_k. The resulting hypothesis is one of the most general discriminating hypotheses that describe the current solution class C_k.

Finally, from the obtained $D = \{d_k^j\}$ where each d_k^j has been simplified, when a simplified d_k^j subsumes another term $d_k^i \in D$ then d_k^i is eliminated from D.

5.4. About INDIE's complexity

The formalism of feature terms is a generalisation of "feature structures" where features may be set-valued. The introduction of sets as values of features increases the expressive power on the formalism in representing relations. This fact effectively allows full relational learning in INDIE. On the other hand, subsumption among "feature structures" is linear on the number of nodes (Carpenter, 1992) while this is not the case for feature terms. The introduction of set-valued features equalises feature-based formalisms with relational representations (like Horn clauses) and the complexity of subsumption now is the same as that of other relational formalisms, e.g. θ-subsumption for ILP.

It is interesting to summarily compare the complexity results known for Horn formalisms with feature terms. Following (Kietz & Lübbe, 1993) we know that θ-subsumption (using Buntine's definition) is NP-complete in general, while $D \sqsubseteq_\theta C$ is polynomial when D is determinate with respect to C. Moreover these authors identify a category of situations, called k-locals, where the worst-case does not apply and they show subsumption for determinate k-local Horn clauses to be polynomial. Another usual provision in ILP is restricting the hypothesis to ij-determinate Horn clauses. As shown in (Lavrac & Dzeroski, 1994) ij-determination depends on the training examples, the background knowledge, and the ordering of literals in a clause. The goal of ij-determination is to assure that there is only one θ-substitution for a clause. Under this hypothesis an efficient treatment of subsumption is achieved since the subsumption complexity arises from the multiplicity of θ-substitutions possible in the general case.

Concerning feature terms, the increase in complexity is clearly caused by the presence of sets of values in a feature. More specifically, the situation where worst case complexity may appear is when feature terms have *embedded sets*. Let $\psi ::= X : s[f_1 \doteq \Psi_1 \ldots f_n \doteq \Psi_n]$ be a term where Ψ_i is a set, and let $\psi' \in \Psi_i$ be a term $\psi' ::= X' : s[f'_1 \doteq \Psi'_1 \ldots f'_n \doteq \Psi'_n]$; if Ψ'_j is a set we say that Ψ'_j is a set *embedded* in set Ψ_i and that ψ has *embedded sets*. Subsumption of two terms with embedded sets in the same features give rise to exponential complexity. Indeed, "feature structures" have linear subsumption because there is only one possible variable substitution from one term to the other (Carpenter, 1992). In feature terms, subsumption among sets of values has to deal with a multiplicity of possible variable substitutions, reintroducing the worst case complexity. When a domain can be represented using feature terms without set-valued features, the subsumption behaves efficiently (in fact, its complexity is lineal).

This fact allows the designer of a particular system to be aware of the complexity introduced when modelling a domain in a certain way: e.g. using the set-valued feature *parents* introduces more complexity than using the *mother* and *father* features. As a further example, while modelling the domains used to evaluate INDIE in Section 6 we found that almost all of them did not require set-valued features; we know thus the complexity we may expect and also that these relational domains were probably designed having in mind the avoidance of the worst case complexity. In our experience, this worst case situation is most of times avoidable (or minimizable) if a domain is modelled carefully.

Concerning anti-unification, the complexity also is caused by the presence of set-valued features and is the same as in other relational formalisms like Horn logic. As before, we know the problems are circumscribed to the features with sets of values and that for the rest of the features the process is efficient. Again, the cause of the complexity is the multiplicity of variable substitutions—present only in set valued features. Anti-unification of "feature structures" is unique while anti-unification of feature terms is not unique: there may be more than variable substitution that gives a most specific subsumer. The multiple variable mappings possible between sets of values is the responsible for the increase of complexity, as before, but is also circumscribed to set-valued features and, the worst case, to the embedded sets case.

6. Evaluation of INDIE

The purpose of this section is to evaluate INDIE showing that is capable to find correct hypotheses for several relational datasets. We have selected datasets commonly used in the

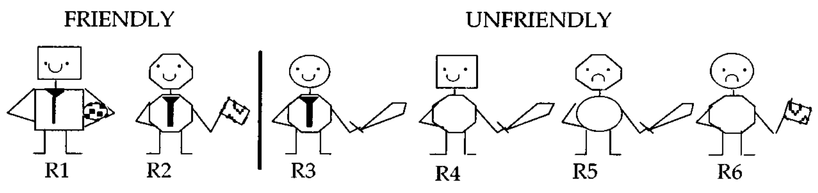


Figure 15. Robots used as input in the Robots dataset.

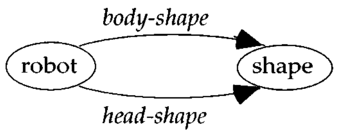
literature to evaluate relational learning systems. We show that INDIE is capable to obtain a correct hypothesis for all them, even for datasets for which some relational learning systems have problems. We also compare INDIE's results with those of the original systems to show how differences in biases and search strategies used by the systems result in finding different correct hypotheses.

6.1. *Robots dataset*

The domain of Robots (Lavrac & Dzeroski, 1994) consists of a description of six robots that belong to two solution classes: *friendly* and *unfriendly* (see figure 15). Each robot is described using five features: `smiling`, `holding`, `has-tie`, `body-shape` and `head-shape`. Robots are described using an attribute-value representation. However, using the feature term formalism, INDIE obtains a relational hypothesis for the *friendly* class:

$$\text{Friendly} = X : \text{robot}\begin{bmatrix} \text{body-shape} \doteq Y : \text{shape} \\ \text{head-shape} \doteq Y : \text{shape} \end{bmatrix}$$

Notice that the value of features `body-shape` and `head-shape` is the same variable Y for sort *shape*. Variable equality, or in other words, path equality, in this hypothesis means that a robot is in the *friendly* class if it has a head whose shape is any shape but is equal to the shape of its body. Path equality is based on the semantics of subsumption ($\sqsubseteq$) in Section 2. Examples R1 and R2 are subsumed by the `Friendly` term above because both have the required path equality; the rest of the examples are not subsumed by the `Friendly` term since they do not satisfy the variable equality constraint present in the subsumer and required by the definition of subsumption. In other words, the *friendly* class hypothesis corresponds to the graph:



The subsumption relation holds only for those examples whose graphs also contain the same path equality (for a particular subsort of `shape`).

LINUS obtains the following rule as description of the *friendly* class:

$$\begin{aligned}\text{Friendly} = {} & [(\text{smiling} = \text{yes}) \wedge (\text{holding} = \text{balloon})] \\ & \vee[(\text{smiling} = \text{yes}) \wedge (\text{holding} = \text{flag})]\end{aligned}$$

Lavrac and Dzeroski (1994) describe how background knowledge can be introduced in attribute-value learning. In addition to attributes describing domain objects, they suggest that the description of domain objects can include attributes representing relations between other attributes. These new attributes are like functions in the sense that their value is *true* or *false*. In particular, descriptions of robots can include a new attribute called `same-shape` that is *true* if both body and head have the same shape and *false* otherwise. According to this new description LINUS obtains the same description for the *friendly* class that INDIE obtains directly.

For the *unfriendly* class INDIE obtains a hypothesis that is a disjunction of two feature terms:

$$\begin{aligned}\text{Unfriendly} = {} & X_1 : \text{robot}\,[\text{has-tie} \doteq \text{no}] \\ & \vee \\ & X_2 : \text{robot}\,[\text{holding} \doteq \text{sword}]\end{aligned}$$

i.e. a robot is *unfriendly* when either it wears no tie or it holds a sword. LINUS using any of learners that it includes (NEWGEM, ASSISTANT or CN2) obtains the following rule:

$$\text{Unfriendly} = [(\text{smiling} = \text{no})] \vee [(\text{smiling} = \text{yes}) \wedge (\text{holding} = \text{sword})]$$

It is worth noting that (smiling = no) plays the same role as the feature [has-tie = no] on INDIE's hypothesis. During INDIE's simplification process, features contained in the not yet simplified hypothesis are ranked according to their relevance using the López de Mántaras distance. For the *unfriendly* class the features `smiling` and `has-tie` have the same value for the distance, and for this reason INDIE randomly chooses one of them to be eliminated. In other words, INDIE can also obtain, with the current biases, the following hypothesis:

$$\begin{aligned}\text{Unfriendly} = {} & X_1 : \text{robot}\,[\text{smiling} \doteq \text{no}] \\ & \vee \\ & X_2 : \text{robot}\,[\text{holding} \doteq \text{sword}]\end{aligned}$$

The other disjunct obtained by LINUS, (smiling = yes) $\wedge$ (holding = sword) is more specific than the one obtained by INDIE. Finally, let us note that introducing the attribute `same-shape`, LINUS obtains that a robot belongs to *unfriendly* class if the attribute `same-shape` has as value *false*, i.e. body and head do not have the same shape.

6.2. Drugs dataset

The domain of Drugs consists of descriptions of several drugs (see in figure 16 some of the descriptions) and has been used by the KLUSTER system (Kietz & Morik, 1994). KLUSTER uses a representation language based on KL-ONE to represent generalisations (class descriptions) and domain objects (instances). From these descriptions KLUSTER

```
Contains(aspirin,asa)             combidrug(anxiolit)
Contains(adumbran, coffein)       combidrug(adolorin)
Contains(adumbran, oxazepun)      monodrug(aspirin)
Contains(anxiolit, oxazepun)      monodrug(adumbran)
Contains(anxiolit, finalin)       active(finalin)
```

Figure 16. Some of the drug descriptions used by the KLUSTER system (Kietz & Morik, 1994).

can classify an instance in one of several classes, i.e. active substance, monodrug, sedative substance, etc. KLUSTER builds hypotheses describing each class (i.e. active substance, monodrug, combidrug, etc.) by searching for a most specific generalisation (MSG) from positive examples. If MSG covers negative examples KLUSTER follows a particular algorithm to specialise MSG by means of introducing new `at-most` and `at-least` predicates in the feature descriptions.

The hypotheses obtained by KLUSTER for *monodrug* and *combidrug* classes are the following:

$$\text{monodrug} := \text{drug and } \textbf{at-least}\ (1, \text{contains-active}) \text{ and } \textbf{at-most}\ (1, \text{contains-active})$$
$$\text{combidrug} := \text{drug and } \textbf{at-least}\ (2, \text{contains-active})$$

In other words, KLUSTER considers that a drug belongs to the *monodrug* class if it contains only one active substance and a drug belongs to the *combidrug* class if it contains at least two active substances.

The hypotheses obtained by INDIE for these classes are similar to those obtained by KLUSTER. The main differences are due to the different representation formalism. INDIE uses anti-unification to find a most specific hypothesis that subsumes positive examples (similarly to KLUSTER since both follow a bottom-up strategy). However, the specialisation in INDIE is achieved by the introduction of a disjunction of hypotheses following the distance-based heuristic. INDIE obtains the following hypothesis for *monodrug* and *combidrug* classes:

$$\text{Monodrug} = X : \text{drug}\begin{bmatrix} \text{effects} \doteq Y : \text{drug-effect} \\ \text{contains} \doteq Z : \text{active-substance}\ [\text{affects} \doteq W : \text{symptom}] \end{bmatrix}$$

$$\text{Combidrug} = X : \text{drug}\begin{bmatrix} \text{contains} \doteq Y : \text{active-substance} \\ \hfill Z : \text{active-substance} \end{bmatrix}$$

The hypothesis of the *monodrug* class means that a substance X is a monodrug when it has some effect Y and it contains an active substance Z affecting some symptom W. A substance X is a *combidrug* when it has two active substances Y and Z. The *combidrug* hypothesis subsumes also a drug with three active substances because of the definition of subsumption (Section 2.1). The reason is that the subsumption definition interprets variables Y and Z as being distinct (i.e. $Y \neq Z$) and requiring thus that the examples to be subsumed have at least two different active substances on the `contains` feature.

This definition of subsumption implies that a simpler *monodrug* hypothesis would also subsume *combidrug* examples—since they have at least one active substance. For this reason

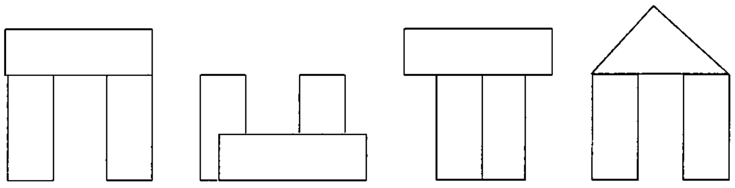


Figure 17. Training examples of the Arch dataset.

INDIE reaches a *monodrug* hypothesis that, in addition to the `contains` feature, has two more discriminating features (`effects` and `affects`).

6.3. *Arch dataset*

Winston (1975) introduced the Arch domain. The Arch Dataset consists of two examples of the *arch* concept and two counter-examples (figure 17). Each arch is composed by three pieces (two verticals and one horizontal). As negative examples, there are two objects also composed of two vertical pieces and one horizontal piece but they do not form an arch. Authors that have used this domain (Quinlan, 1990; Lavrac & Dzeroski, 1994) represent background relations as Horn clauses.

The arch hypothesis obtained by LINUS and FOIL (both using the closed world assumption) is the following:

$$\text{arch}(A, B, C) \leftarrow \text{left-of}(B, C), \text{supports}(B, A), \text{not touches}(B, C)$$

The predicate `supports` means that a block has another block on top of it; the predicate `touches` means that a block has a lateral contact with a second block; and he predicate `over` means that a block is on top of another block. Lavrac and Dzeroski (1994) noticed that the hypothesis obtained by LINUS and FOIL is correct with respect to the positive and negative examples in figure 17 but it is not correct because it covers unseen examples such as those in figure 18 that are not arches.

Thus, LINUS and FOIL need to include both objects as negative examples in order to obtain a correct description of *arch*. However, using the training set of figure 17, INDIE

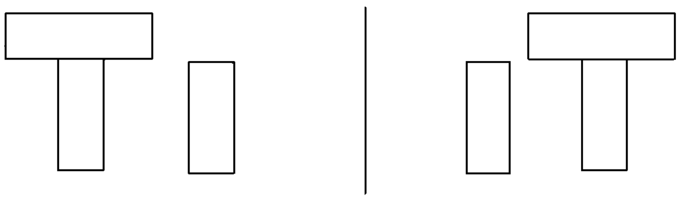


Figure 18. Two negative examples of *arch* covered by the description obtained by FOIL and LINUS.

obtains the following hypothesis that does not subsume the unseen examples in figure 18:

$$\text{Arch} = X : \text{figure}\left[\text{left} \doteq Y : \text{brick}\left[\begin{array}{l} \text{left-to} \doteq T : \text{brick}\left[\begin{array}{l}\text{right-to} \doteq Y \\ \text{supports} \doteq Z \\ \text{touches} \doteq \text{no-one}\end{array}\right] \\ \text{supports} \doteq Z : \text{block}\left[\text{over} \doteq \begin{array}{c} T \\ Y \end{array}\right] \\ \text{touches} \doteq \text{no-one} \end{array}\right]\right]$$

This hypothesis states that an *arch* is an object X having a `brick` Y (`brick` is a subsort of `block`) satisfying the following conditions: (1) Y is left to a brick T, (2) Y supports a block Z, and (3) Y does not touch any brick (`touches` feature has value `no-one`). In turn, the block Z has to be over bricks T and Y. Finally, brick T is to the right of Y, supports block Z and does not touch any brick. Notice that the hypothesis above obtained by INDIE does not subsume the unseen negative examples in figure 18, since both vertical bricks have to support the central block Z.

6.4. *Families dataset*

This domain, defined by Hinton (1989), consists of the definition of two families having twelve members each (see figure 19). Several relational learning systems have been tested using this domain. In particular, LINUS obtains the following hypotheses for the *mother* relation:

$$\text{mother}(A, B) \leftarrow \text{daughter}(B, A),\ \text{not father}(A, B) \tag{1}$$
$$\text{mother}(A, B) \leftarrow \text{son}(B, A),\ \text{not father}(A, B) \tag{2}$$

The rules obtained by FOIL to describe the mother relation are the following:

$$\text{mother}(A, B) \leftarrow \text{daughter}(B, A),\ \text{not father}(A, C) \tag{3}$$
$$\text{mother}(A, B) \leftarrow \text{son}(B, A),\ \text{not father}(A, C) \tag{4}$$

Notice that FOIL obtains more specific hypothesis than LINUS since (3) and (4) use a new variable C. This new variable means that "A is not the father of anybody", whereas in

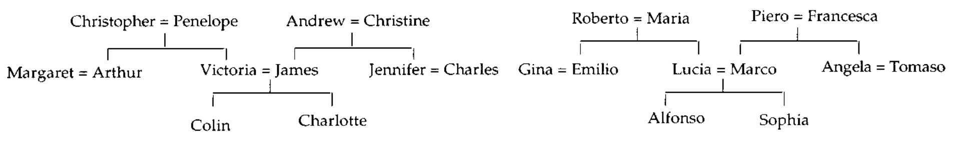


Figure 19. Training set of the Families dataset.

descriptions (1) and (2) "A is not the father of B" (i.e. A could be the father of a person different than B).

INDIE obtains the following hypothesis for mother:

$$\text{mother} = X : \text{female}\,[\text{son} \doteq Y : \text{male}]$$

This hypothesis is equivalent to descriptions (2) and (4) above since the relation *not father* (used by LINUS and FOIL) is equivalent to defining a person of sort *female* as INDIE does (both rule out the same collection of examples). Notice that including in the hypothesis only the son feature (and not the daughter feature) is sufficient since in the training set all mothers have one son—and, also, one daughter. INDIE obtains only one disjunct because the hypothesis obtained by anti-unification already subsumes all positive examples and does not subsume any negative example—thus no specialisation step was needed. After the simplification post-process only the son feature remains. During the post-process, two features, son and daughter, have the same RLM distance; because of this any of them could have been eliminated, and INDIE has randomly chosen the elimination of the daughter feature.

Using INDIE to obtain a hypothesis for class *uncle* the result is the following:

$$\text{uncle} = X : \text{male}\,[\text{niece} \doteq Y : \text{female}]$$

That is to say, an uncle is a male that has (at least) a niece. As before, during the simplification post-process, two features, niece and nephew, have the same RLM distance, therefore INDIE has randomly chosen the elimination of the nephew feature.

6.5. *Trains dataset*

This domain was introduced by Michalski (1980) to test the INDUCE system. Domain objects are 10 trains (see figure 20) having different numbers of cars with various shapes carrying loads of different forms.

The task is to distinguish between *eastbound* and *westbound* trains. Learners that use attribute-value representation have a great difficulty to solve this task due to the variability in the number of structures and substructures present in the domain objects (e.g. trains have a variable number of cars). LINUS needs the introduction of an artificial variable (namely, the number of cars) to obtain the *eastbound* description (see Lavrac, Dzeroski, & Grobelnik, 1991). Using ASSISTANT, LINUS obtains a hypothesis consisting of 19 Prolog clauses that after post-processing is reduced to one clause that is the same obtained by FOIL and INDUCE:

$$\text{eastbound}(A) \leftarrow \text{has-car}(A, B), \neg\,\text{long}(B), \neg\,\text{open-top}(B)$$

Concerning westbound trains, we have no information about the LINUS results. The FOIL system obtains the following hypothesis for the *westbound* trains:

$$\text{westbound}(A) \leftarrow \text{has-car}(A, B),\ \text{long}(B),\ \text{2- wheels}(B), \neg\,\text{open-top}(B)$$

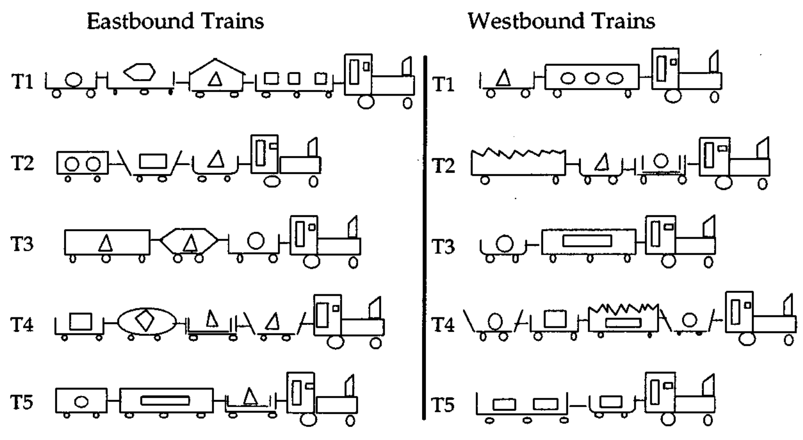


Figure 20. Training examples of the Trains Dataset. In this training set, the solution classes are *eastbound* and *westbound* trains.

This hypothesis covers three of the five westbound trains. FOIL is not capable of obtaining a complete hypothesis due to the encoding length heuristics. INDUCE can obtain a hypothesis covering all the westbound trains because it uses constructive induction that introduces a new predicate: the number of cars of a train. Thus, the *westbound* class is described by INDUCE as follows:

$$\text{westbound}(A) \leftarrow \text{car-count}(A) = 3$$
$$\text{westbound}(A) \leftarrow \text{has-car}(A, B),\ \text{jagged-top}(B)$$

Using feature terms INDIE avoids the problem of the variability in the number of structures and substructures and is capable of obtaining a hypothesis for both *eastbound* and *westbound* trains without introducing new predicates. The *eastbound* hypothesis obtained by INDIE is the disjunction of the following term:

$$\text{eastbound} = X_1 : \text{train [wagons} \doteq Y_1 : \text{closed-car [length} \doteq \text{short]]}$$

i.e. the eastbound trains are characterised by having one wagon that is a closed car which is short. This description of the eastbound trains is equivalent to that obtained by FOIL and INDUCE. Notice that INDIE obtains that the wagon is short whereas FOIL and INDUCE obtain the negation of the predicate “long”.

The *westbound* hypothesis obtained by INDIE is also a disjunction of three feature terms:

$$\begin{aligned}\text{westbound} = {} & X_1 : \text{train [wagon2} \doteq Y_1 : \text{open-car [form-car} \doteq \text{openrect]]}\\ & \vee\\ & X_2 : \text{train [wagon2} \doteq Y_2 : \text{open-car [from-car} \doteq \text{ushaped]]}\\ & \vee\\ & X_3 : \text{train [wagon3} \doteq Y_3 : \text{open-car [load-set} \doteq \text{rectanglod]]}\end{aligned}$$

Now the `form-car` feature, in addition to the `load-set` feature, are the most relevant to describe the *westbound* class.

6.6. Discussion

INDIE is capable of inducing correct hypotheses for relational domains commonly used in the literature. This section shows that INDIE provides correct results in robots, drugs, families and arch domains and that they are comparable to those obtained by FOIL, KLUSTER, INDUCE and LINUS. Notice that in the Robots domain INDIE obtains a relational hypothesis for *friendly* class (i.e. a robot belongs to the *friendly* class if it has the same shape of head and body). This same hypothesis is obtained by LINUS only after manually introducing a new predicate representing the `same-shape` relation. INDIE also obtains better results than FOIL, INDUCE and LINUS when it is applied to the arch domain, since INDIE does not need additional negative examples to find a correct description of arch. Results obtained by INDIE over the trains domains are comparable albeit different from those obtained by FOIL, INDUCE and LINUS. One reason for the different description for *eastbound* class is that INDIE does not use negation. On the other hand, both FOIL and INDUCE systems have some difficulties in obtaining a description for the *westbound* class: FOIL cannot obtain a description covering the five westbound trains and INDUCE has to introduce a new predicate in order to achieve it. Instead, INDIE obtains a hypothesis composed of the disjunction of three terms for *westbound* class without needing predicate invention.

7. Application of INDIE to the marine sponges domain

The identification of specimens is a very common task in biological research. There are several types of biological studies that need a taxonomic analysis of organisms, for instance environmental studies. Frequently, an error in the identification of the organisms invalidates the whole study. The identification of marine sponges is especially complex and often the support of an expert is necessary. Moreover, sponges are genetically much more diverse than other marine invertebrates. They also have high variability in form within a species due to their plastic ability to adapt to environmental conditions. As an example of the complexity in the identification of marine sponges, we want to remark that some researchers have assumed the discovery of a new species whereas it was really a morph of an already known species. The taxonomic classification of any group of organisms has the aim of establishing a hierarchical organisation of taxa.

Commonly, the identification of specimens is made from the descriptions of the taxa. Therefore, given a sponge specimen, a strategy for identification is to explore the taxonomy and to find, for each taxonomy level, one taxon in which this new specimen can be classified. To follow this strategy, it is necessary to known the taxa descriptions, but there is no agreement among the experts about which are the characteristic features of the taxa. Nevertheless, there are many sponges whose identification is not discussed, so they could be used to identify new sponges. Our purpose is to use an inductive learning method like INDIE to induce the descriptions of the taxa from the already classified sponges and later verify the obtained hypotheses with an expert marine biologist. Notice that the sponge

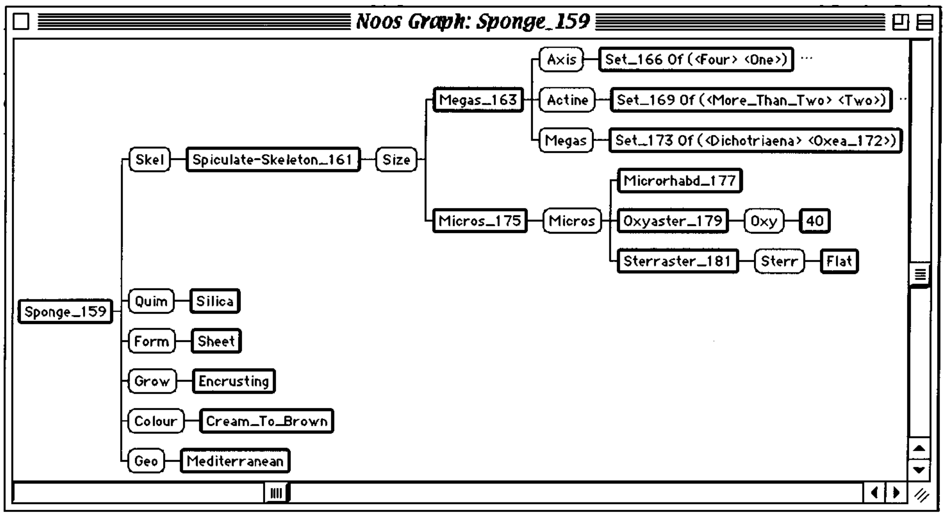


Figure 21. Browser showing a description of a sponge specimen.

identification process is a multi-layered classification task since each specimen is classified in five taxonomic levels: *class*, *order*, *family*, *genus* and *species*. Consequently, we need to apply INDIE to each of these levels in order to achieve a discriminating description for the classes (taxa) of each level.

In this section we explain the use of INDIE to obtain hypothesis characterising genus of sponges. A sponge is represented by a feature term of root sort *sponge*, an example of which is shown in figure 21. Most descriptions of sponges in our dataset are not complete, i.e. often values of relevant features are unknown. The reason for having partial descriptions lies in the incompleteness of the original reports on sponge specimens, often due to the fact that biologist work with pieces of sponges.

In the experiment reported here we use a training set containing 26 marine sponges correctly classified in the *genus* level. At this level, the sponges can belong to one of five genus: *caminus*, *erylus*, *isops*, *pachymatisma* and *geodia*. INDIE has been used to find a discriminating hypothesis for each genus. Let us to analyse the obtained hypothesis.

There are three specimens in the training set belonging to genus *caminus*. Two of these have a detailed description (around 12 features) whereas the other is described using only 3 features, a clear case of partial information. Figure 22 shows the feature term that INDIE (with the simplification post-process) has built for the *caminus* genus. The hypothesis states

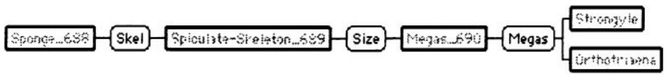


Figure 22. Hypothesis obtained by INDIE (with the simplification post-process) for the *caminus* genus.

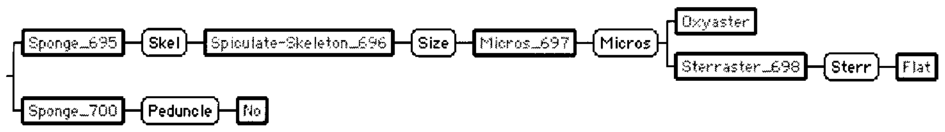


Figure 23. Browser showing a disjunct of two feature terms obtained for the genus *erylus*.

that the skeleton is of sort spiculate, in which there are big spicules (called megascleres, shown as the sort *megas* in figure 22), and there are two kinds of megascleres (shown as the `megas` feature), namely strongyle and orthotriaena.

Nine of the specimens in the training set belong to genus *erylus*. Each specimen is described usually by around 6 features, but few features are common to all them. The first step of INDIE obtains a first hypothesis by anti-unification. Since it is not discriminating, INDIE specialises it and then finds the disjunction of two terms in figure 23. One of these terms states that the skeleton is of sort spiculate, in which there are small spicules (called microscleres), of two kinds (oxyaster and sterraster), and where the form of this last one is flat. The second term states that a specimen is *erylus* when it has no peduncle.

The training set has 8 sponges belonging to the genus *geodia*. These sponges are described by a number of features varying between 5 to 10. This variability among the descriptions accounts for the difficulty in finding a set of discriminating features common to all the *geodia* specimens. Thus, INDIE needs several specialisations of the hypothesis obtained from the anti-unification of the specimens of *geodia* in order to build a discriminating description. Figure 24 shows the disjunction of 5 terms obtained by INDIE to describe the genus *geodia*.

There are only two specimens in the training set that belong to the genus *pachymatisma*. Both specimens have very similar descriptions. In fact they only differ in that the *spiculate-skeleton* term in one specimen has a feature called `spicarch` whereas this feature is not present in the other one. The hypothesis obtained from the anti-unification of these two specimens has the same features and values as the description of the second one. Since this hypothesis does not subsume negative examples it is correct. After the simplification

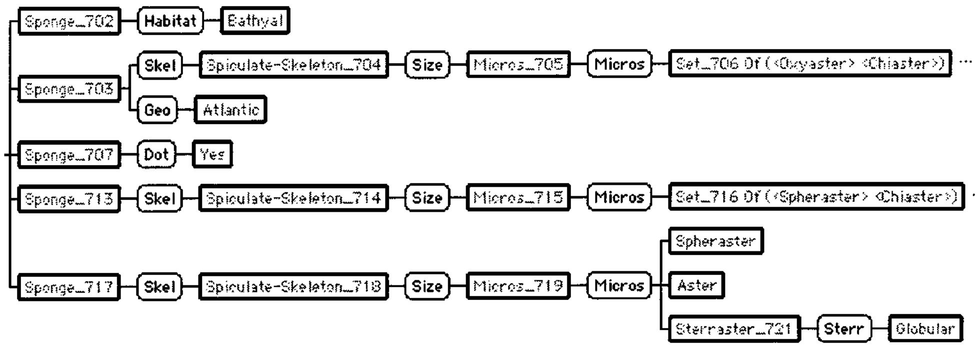


Figure 24. Disjunction of 5 terms for the genus *geodia*.

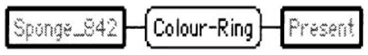


Figure 25. Feature term obtained for the genus *pachymatisma*.

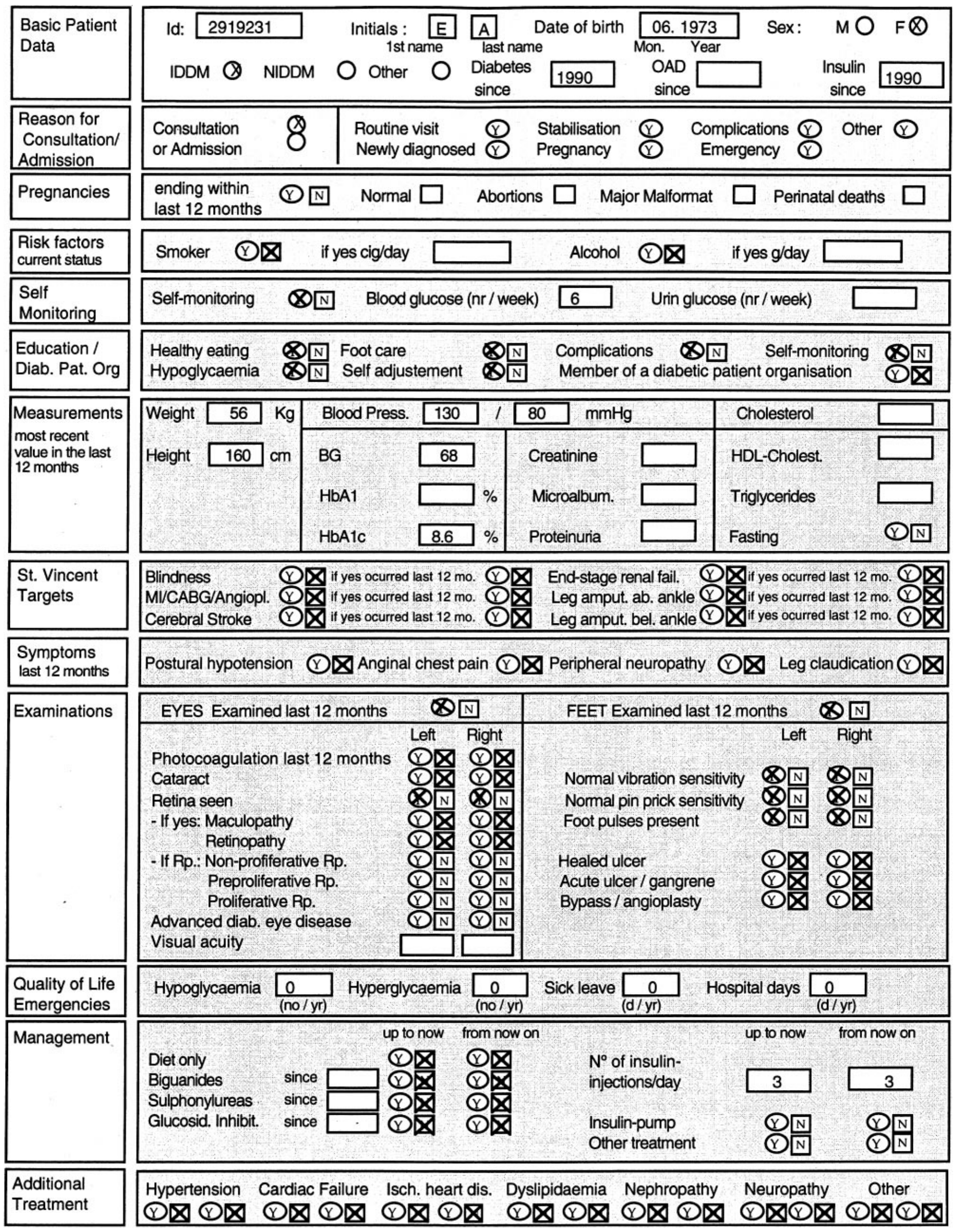
post-process the hypothesis is the one shown in figure 25. This hypothesis characterises the *pachymatisma* genus by the presence of a coloured ring that is, in fact, the feature giving name to the genus. This example shows the utility of the heuristic used in the simplification post-process, eliminating the less discriminating features, since the most discriminating feature is the one remaining in the simplified hypothesis.

The descriptions obtained using INDIE have been presented to a marine biologist expert in this domain that has considered these descriptions to be accurate. However, the focus of INDIE, as most other relational learners, is in finding the simplest hypothesis that is correct. For instance the description provided by INDIE for genus *pachymatisma* (figure 25) is very short (it contains only the feature `colour-ring`). Our expert, in the task of *explaining* a genus, would provide descriptions having more features; in fact, she tended to produce a “prototype”-based description—i.e. a description based on the most common (typical) values. Nonetheless, after some case studies, she agreed that the descriptions obtained by INDIE with the simplification post-process were not only correct but they also contained those essential features in which a expert focuses her attention to classify a specimen, that is to say for the task of *identifying* the genus of a sponge (Domingo, personal communication).

8. Application of INDIE to the diabetes domain

This section shows an application of INDIE to learn to discriminate among levels of risk incurred by diabetic patients. Diabetes is a metabolic disease characterised by hyperglycaemia and the short-term and long-term symptoms related to it. Long-term symptoms are the more important ones since they affect the life quality of the diabetic patient. The chronic nature of diabetes is associated with the risk of developing *complications* (like blindness and vascular problems) and, once the complication is developed, with the risk of *progression* of their dangerous effects. In turn, these complications depend on the diabetes evolution and on the hyperglycaemia degree.

INDIE’s application’s goal is, for each associated complication, to induce a class hypothesis for each level of risk—namely `unknown`, `low`, `moderate`, `high`, `very-high` both for (1) risk of developing a complication and (2) risk of progression for a developed complication. In particular we will show examples for two complications: (a) global macrovascular complications and (b) stroke, a specific macrovascular complication. INDIE’s learning is performed over records of diabetic patients taken from the DiabData database. DiabData contains the data considered as necessary for the clinical purposes defined in the DiabCare project. DiabData patient records are currently gathered in 17 European countries on the basis of the St. Vincent Declaration, a document agreed by representatives of Government Health Departments, patient organisations, and diabetes experts from all European countries. In this document they agreed upon the recommendations about the diabetes treatment (more information is online at `http://diabcare.de/dimeu.html`). Figure 26 shows the basic information sheet of a diabetic patient in the DiabData format.



Basic Patient Data	Id: 2919231 · Initials: E (1st name) A (last name) · Date of birth 06. 1973 (Mon. Year) · Sex: M [✓ F]
	[✓ IDDM] NIDDM Other · Diabetes since 1990 · OAD since · Insulin since 1990
Reason for Consultation/ Admission	[✓ Consultation] or Admission · Routine visit Y · Newly diagnosed Y · Stabilisation Y · Pregnancy Y · Complications Y · Emergency Y · Other Y
Pregnancies	ending within last 12 months Y N · Normal · Abortions · Major Malformat · Perinatal deaths
Risk factors current status	Smoker Y [✓ N] if yes cig/day · Alcohol Y [✓ N] if yes g/day
Self Monitoring	Self-monitoring [✓ Y] N · Blood glucose (nr / week) 6 · Urin glucose (nr / week)
Education / Diab. Pat. Org	Healthy eating [✓ Y] N · Foot care [✓ Y] N · Complications [✓ Y] N · Self-monitoring [✓ Y] N
	Hypoglycaemia [✓ Y] N · Self adjustement [✓ Y] N · Member of a diabetic patient organisation Y [✓ N]
Measurements most recent value in the last 12 months	Weight 56 Kg · Height 160 cm
	Blood Press. 130 / 80 mmHg · BG 68 · HbA1 % · HbA1c 8.6 %
	Creatinine · Microalbum. · Proteinuria
	Cholesterol · HDL-Cholest. · Triglycerides · Fasting Y N
St. Vincent Targets	Blindness Y [✓ N] if yes ocurred last 12 mo. Y [✓ N] · End-stage renal fail. Y [✓ N] if yes ocurred last 12 mo. Y [✓ N]
	MI/CABG/Angiopl. Y [✓ N] if yes ocurred last 12 mo. Y [✓ N] · Leg amput. ab. ankle Y [✓ N] if yes ocurred last 12 mo. Y [✓ N]
	Cerebral Stroke Y [✓ N] if yes ocurred last 12 mo. Y [✓ N] · Leg amput. bel. ankle Y [✓ N] if yes ocurred last 12 mo. Y [✓ N]
Symptoms last 12 months	Postural hypotension Y [✓ N] · Anginal chest pain Y [✓ N] · Peripheral neuropathy Y [✓ N] · Leg claudication Y [✓ N]

Examinations

EYES Examined last 12 months [✓ Y] N	Left	Right
Photocoagulation last 12 months	Y [✓ N]	Y [✓ N]
Cataract	Y [✓ N]	Y [✓ N]
Retina seen	[✓ Y] N	[✓ Y] N
- If yes: Maculopathy	Y [✓ N]	Y [✓ N]
Retinopathy	Y [✓ N]	Y [✓ N]
- If Rp.: Non-profiferative Rp.	Y N	Y N
Preproliferative Rp.	Y N	Y N
Proliferative Rp.	Y N	Y N
Advanced diab. eye disease	Y N	Y N
Visual acuity		

FEET Examined last 12 months [✓ Y] N	Left	Right
Normal vibration sensitivity	[✓ Y] N	[✓ Y] N
Normal pin prick sensitivity	[✓ Y] N	[✓ Y] N
Foot pulses present	[✓ Y] N	[✓ Y] N
Healed ulcer	Y [✓ N]	Y [✓ N]
Acute ulcer / gangrene	Y [✓ N]	Y [✓ N]
Bypass / angioplasty	Y [✓ N]	Y [✓ N]

Quality of Life Emergencies				
	Hypoglycaemia 0 (no / yr)	Hyperglycaemia 0 (no / yr)	Sick leave 0 (d / yr)	Hospital days 0 (d / yr)

Management	since	up to now	from now on
Diet only		Y [✓ N]	Y [✓ N]
Biguanides		Y [✓ N]	Y [✓ N]
Sulphonylureas		Y [✓ N]	Y [✓ N]
Glucosid. Inhibit.		Y [✓ N]	Y [✓ N]

	up to now	from now on
N° of insulin-injections/day	3	3
Insulin-pump	Y N	Y N
Other treatment	Y N	Y N

Additional Treatment	Hypertension	Cardiac Failure	Isch. heart dis.	Dyslipidaemia	Nephropathy	Neuropathy	Other
	Y [✓ N] Y [✓ N]	Y [✓ N] Y [✓ N]	Y [✓ N] Y [✓ N]	Y [✓ N] Y [✓ N]	Y [✓ N] Y [✓ N]	Y [✓ N] Y [✓ N]	Y [✓ N] Y [✓ N]

Figure 26. Data of a diabetic patient as it is agreed in the Saint Vincent declaration.

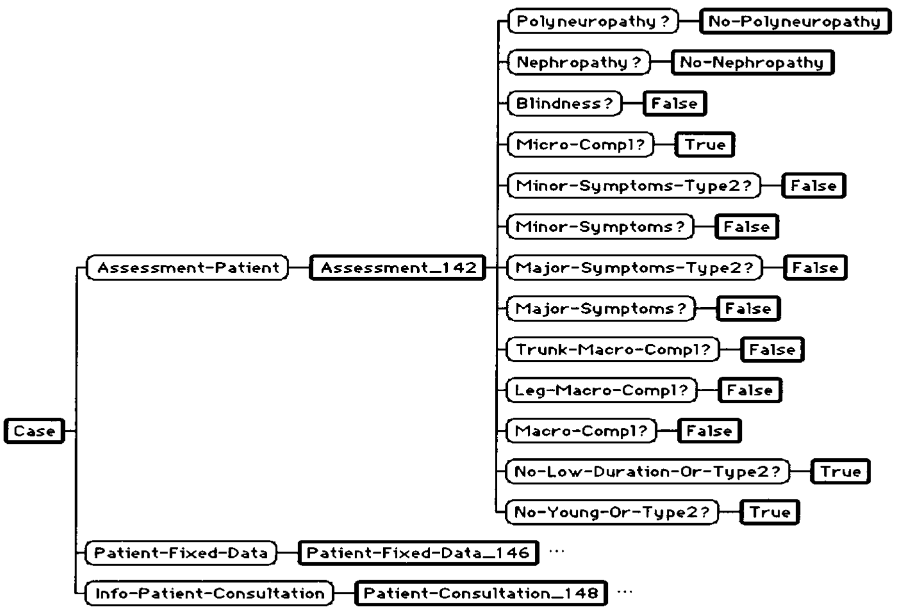


Figure 27. Representation of a diabetic patient using feature terms.

We have translated the data in DiabData format into the feature term representation shown in figure 27. A patient *case* has three kinds of information: personal information (`patient-fixed-data`), information gathered during a consultation (`info-patient-consultation`), and a brief assessment of the patient situation (`assessment-patient-db`). The two first kinds hold data coming from DiabData database (see the information sheet in figure 26). Moreover, the patient situation assessment is a brief summary that is shown to the doctor to help him in determining the patient risks.[1] The patient situation assessment transforms numerical data into meaningful qualitative values, assess changes between the last consultation and the present one, and summarises which complications are presented by a patient—but does not determine or assess any kind of risk for the patient. For instance the `leg-macro-compl?` feature (see figure 27) ascertains the presence or absence of macrovascular complications on the legs of a patient while `Blindness?` ascertains whether or not the patient is considered blind.

We have applied INDIE to 100 cases containing the aforementioned data for estimating different patient risks: macro and microvascular complications, stroke, amputation, blindness, nephropathy, and polyneuropathy. We will presently show two examples of the induced hypotheses, one for progression risk and another for development risk.

Concerning patients with stroke complication we are interested in learning discriminating hypotheses for the different levels of progression risk—namely `unknown`, `low`, `moderate`, `high`, `very-high`. Let us now consider the class of patients with low progression risk: after furnishing INDIE with the patients in this class as positive examples and the rest as negative examples the hypothesis obtained is that of figure 28. This hypothesis states that

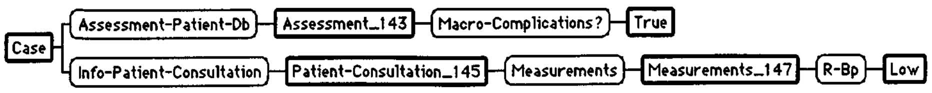


Figure 28. Hypothesis characterising the class of diabetic patients with a low risk of stroke progression.

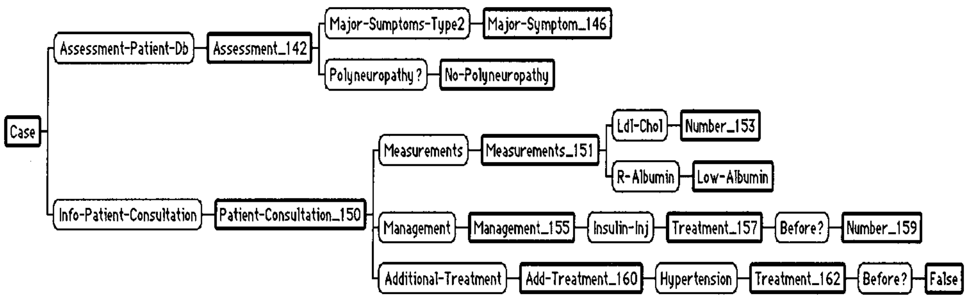


Figure 29. Hypothesis characterising the class of diabetic patients with a low development risk of global macrovascular complications.

progression risk of stoke is low when macrovascular complications are present and the blood pressure (`R-Bp`) is low. Similar hypotheses were found for the classes of moderate and high risk in which blood pressure (`R-Bp`) is respectively moderate and high. These hypotheses are correct with respect to the 100 training patients used and corroborated by the expert diabetes doctor since it is known that blood pressure is highly correlated to stroke risk for these kind of patients.

Let us now consider the class of patients that may develop global macrovascular complications in the future. In particular, for those patients whose development risk is *low* the hypothesis obtained by INDIE is shown in figure 29. The hypothesis states that the future risk of developing macrovascular complications is low for patients that (1) have a known value for major symptoms of type-2 diabetes, (2) do not present the polyneuropathy complication, (3) have a known value for LDL-cholesterol, (4) have a low value of albumin, (5) are insulin-dependent (because he's been treated with insulin since a certain date indicated by the number in the feature `before?`), and (6) have not had any treatment for hypertension (indicated by the `false` value of the feature `before?` of hypertension treatment).

In two places above, in order to describe the hypothesis, we have said that a "value is known", like when we said that it has a known value for major symptoms of type-2 diabetes. In fact, we mean that feature `major-symptoms-type2` has a value of sort *major-symptom*; since this sort has some subsorts (long diabetes duration, moderate or high cholesterol, high blood pressure, age higher than 55, high body-mass index, and smoking) what the hypothesis reveals is that all the positive examples have (at least) one of these symptoms. Similarly, the value of LDL-cholesterol being of sort number reveals that some numeric value is known for all positive examples.

9. Conclusions

INDIE is a heuristic bottom-up inductive learning method that uses feature terms to represent domain objects and the hypotheses it builds. We have seen on several common relational problems that INDIE is capable of finding hypotheses at a level comparable to FOIL, LINUS and INDUCE. The differences between INDIE's hypotheses and those of the other systems are explained by the bias in searching the hypothesis space and on the representational bias of the hypothesis language of each system. The representational bias of INDIE can be summarised in that it makes an intensive use of sorts and sort hierarchy, and in that it does not use negation but focuses on detecting path equalities.

Path equality, embodying variable equality in the hypothesis space, is a powerful construct for capturing regularities in data, and is one of the main properties of feature terms (Carpenter, 1992). A precedent, although not direct, of path equality for Machine Learning is the relational pathfinding technique (Richards & Mooney, 1992). Relational pathfinding explores however a smaller hypothesis space than INDIE since it can only find hypotheses expressible as combinations of path equalities. However, it is a distinct, ad hoc technique (to be included in a wider learning system like Richards and Mooney's FORTE system) while in the framework of feature terms path equality is part and parcel of the representation scheme, present in the definitions of subsumption and anti-unification.

A second precedent is KLUSTER, an inductive system for description logics that we have commented on Section 6.2 for the drugs dataset, the only one reported to be used (Kietz & Morik, 1994). Both KLUSTER and INDIE follow a bottom-up strategy (thus using anti-unification from positive examples) but KLUSTER's overgeneral hypothesis are specialised by introducing new specific predicates (`at-most` and `at-least`) while INDIE can use any predicate (feature) and chooses the one selected by the RLM distance heuristic. KLUSTER uses a KL-ONE-like representation where it is known that introducing path equality together with the usual description logic constructs (like negation and cardinality constraints) increases the complexity of subsumption to intractability. On the other hand, feature terms do not incorporate negation and it focuses around the notion of path equality—nonetheless it is able to deal with the datasets taken from relational learning literature and shown in Section 6. The introduction of sets as values of features results in the expected increase in complexity; however, this is a worst case bound and, for the datasets in Section 6, there has been no problems in tractability in spite of a heavy use of the subsumption operation. As a result, we know when feature term representation can be efficiently used for learning in domains where relational representation is adequate: for domains we can model with no set valued features—since then subsumption is linear—or few set valued features—since then the complexity is manageable in practice.

Acknowledgments

This work has been developed in the context of the SMASH project supported by the Spanish Project CICYT TIC96-1038-C04-01. Authors thank Marta Domingo, Albert Palaudàries and Ana M^a Monteiro for their collaboration in developing marine sponges and diabetes applications.

Note

1. The patient situation assessment module uses expert domain knowledge and has been developed by Ana M^a Monteiro in the context of the Systems of Multi-agents for Medical Services in Hospitals (SMASH) project.

References

Armengol, E. & Plaza, E. (1998). INDIE: A bottom-up method inducing feature terms. IIIA Research Report.

Aït-Kaci, H. & Podelski, A. (1993). Towards a meaning of LIFE. *Journal Logic Programming, 16*, 195–234.

Carpenter, B. (1992). The logic of typed feature structures. *Tracts in Theoretical Computer Science*. Cambridge, UK: Cambridge University Press.

Hinton, G. E. (1989). Connectionist learning procedures. *Artificial Intelligence, 40*, 185–234.

Kietz, J. U. & Lübbe, M. (1993). An efficient subsumption algorithm for inductive logic programming. In *Proceedings of the Tenth International Conference on Machine Learning* (pp. 130–138).

Kietz, J. U. & Morik, K. (1994). A polynomial approach to the constructive induction of structural knowledge. *Machine Learning, 14*, 193–217.

Lavrac, N. & Dzeroski, S. (1994). *Inductive Logic Programming*. London: Ellis Horwood Limited. Ellis Horwood Series in Artificial Intelligence.

Lavrac, N., Dzeroski, S., & Grobelnik, M. (1991). Learning non-recursive definitions of relations with LINUS. In *Lecture Notes in Artificial Intelligence*, Vol. 482: *Proceedings of the 5th European Working Session on Learning*. Y. Kodratoff (Ed.). Springer-Verlag.

López de Mántaras, R. (1991). A distance-based attribute selection measure for decision tree induction. *Machine Learning*, *6*, 81–92.

Michalski, R. S. (1980). Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *2*, 349–361.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill. McGraw Hill Series in Computer Science.

Muggleton, S. & De Raedt, L. (1994). Inductive logic programming: theory and methods. *Journal of Logic Programming*, *19*, *20*, 629–679.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 5*, 239–266.

Richards, B. L. & Mooney, R. J. (1992). Learning relations by pathfinding. In *Proceedings of the AAAI* (pp. 50–55).

Winston, P. H. (1975). Learning structural descriptions from examples. In P. H. Winston (Ed.), *The Psychology of Computer Vision*. New York: McGraw-Hill.

Received October 26, 1998
Accepted March 20, 2000
Final manuscript March 20, 2000