Part II:
Regular Papers

Induction of Feature Terms With INDIE

Eva Armengol and Enric Plaza

IIIA - Institut d'Investigació en Intel·ligència Artificial
CSIC - Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia, Spain.
Vox: +34-3-5809570, Fax: +34-3-5809661
{eva,enric}@iiia.csic.es
http://www.iiia.csic.es

Abstract. 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. Feature terms have been studied to formalize object-centered representation in declarative languages and can be seen as a subset of first-order logic. We present a representation formalism based on feature terms and we show how induction can be performed in a natural way using a notion of subsumption based on an informational ordering. Moreover feature terms also allow to specify incomplete information in a natural way. An example of such inductive methods, INDIE, is presented. INDIE performs bottom-up heuristic search on the subsumption lattice of the feature term space. Results of this method on several domains are explained.

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 learning research is thus biased by the representation formalism used. Most work on ILP (inductive logic programming) has been focused on induction in subsets of first order logic like Horn or Datalog clauses. We think ML research can also profit from exploring other representation formalisms that allow the expressive power of relations but are different subsets of fist order logic.

In this paper we present a representation formalism based on feature terms and INDIE, a bottom up learning method that induces class descriptions in the form of feature term from positive and negative examples. Feature terms (also called feature structures or *ψ-terms*) are a generalization of first-order terms that have been introduced in theoretical computer science in order to formalize object-oriented capabilities into declarative languages. Feature term formalisms have a family resemblance with—but are different from—unification grammars and description logics (KL-One-like languages) [1, 5].

An advantage of feature terms is that they allow a natural way to describe incomplete information [1]. Incomplete information arises the so-called problem of "unknown values" in ML and, specially in attribute-value representation, the

problem of irrelevant attributes. For instance, in § 4 we present the induction of class descriptions for the identification of marine sponges using INDIE. It turns out that depending on the kind of skeleton a sponge may have (*fiber* or *spiculate*) a collection of attributes is irrelevant to the description of the sponge. Sorts (types) in feature terms solve this problem by relating each predicate (attribute) to the sort to which it is relevant. Following our example, the *spicarch* predicate is relevant to skeletons of sort *spiculate* but not to skeletons of sort *fiber*.

Feature terms form a lattice by means of the subsumption relationship. From subsumption (equivalent to the *more general than* relation in ML) it is natural to define the operations of unification and anti-unification (AU) in which INDIE is based (see §2). While the generalization relation is the natural one for induction, and has been used extensively in classical ML methods, most relational learners based on Horn clauses are based on some notions of inversion deduction (resolution, entailment, etc). In these ILP approaches the generalization relation has to be derived from deduction and, since there are several ways in which this can be done, the generalization relation has been a focus of research and debate—see [4] for a thorough summary of the different proposals.

The structure of the paper is the following. First, feature terms are formally described and then subsumption and AU operations are defined. Then §3 presents an inductive method for feature terms INDIE, based on the subsumption and AU operations. § 4 shows the results of INDIE in several domains—including standard ML data sets. Finally, related work and our final conclusions are discussed.

2 Feature Terms

Feature terms are a way to construct terms. 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, variables are identified by position. The intuition behind a feature term is that it can be described as a labeled graph—in other words, variables are identified by name (regardless of order or position). This difference allows to represent partial knowledge.

Feature terms are just terms and require to be integrated into a representation formalism to be used in representation, reasoning and learning. We will presently introduce the role of feature terms in the reflective object-centered representation language Noos [2]. Noos was designed to support the integration of learning methods into knowledge modeling frameworks and here we will merely present the subset needed to explain induction of descriptions from examples. Intuitively, Noos extends the formalisms of [1, 5] allowing the values of features to be *sets of values*.

2.1 Feature Terms in Noos

Our approach to formalize Noos is related to the research based on *ψ-terms* [1, 5], and extensible records [7] that propose formalisms to model object-oriented pro-

gramming constructs. Noos is an object-centered representation language based on *feature terms*. *Feature terms* are record-like data structures embodying a collection of *features*.

We describe the Noos signature Σ as the tuple $\langle \mathcal{S}, \mathcal{F}, \leq \rangle$ such that:

- $\mathcal{S}$ is a set of *sort symbols* including $\bot, \top$;
- $\mathcal{F}$ is a set of *feature symbols*;
- $\leq$ is a decidable partial order on $\mathcal{S}$ such that $\bot$ is the least element and $\top$ is the greatest element.

We define an interpretation $\mathcal{I}$ over the signature $\langle \mathcal{S}, \mathcal{F}, \leq \rangle$ as the structure

$$\mathcal{I} = \langle \mathcal{D}^{\mathcal{I}}, (f^{\mathcal{I}})_{f \in \mathcal{S}}, (\ell^{\mathcal{I}})_{\ell \in \mathcal{F}} \rangle$$

such that:

- $\mathcal{D}^{\mathcal{I}}$ is a non-empty set, called *domain* of $\mathcal{I}$ (or, universe);
- for each symbol s in $\mathcal{S}$, $s^{\mathcal{I}}$ is a subset of the domain; in particular, $\top^{\mathcal{I}} = \mathcal{D}^{\mathcal{I}}$ and $\bot^{\mathcal{I}} = \emptyset$;
- for each feature ℓ in $\mathcal{F}$, $\ell^{\mathcal{I}}$ is a total unary function $\ell^{\mathcal{I}} : \mathcal{D}^{\mathcal{I}} \mapsto \mathcal{P}(\mathcal{D}^{\mathcal{I}})$. When the mapping is not defined it is assumed to have value $\top$.

Given the signature Σ and a set $\mathcal{V}$ of variables, we define feature terms as:

Definition 1. A feature term ψ is an expression of the form:

$$\psi \quad ::= \quad X : s\,[f_1 \doteq \Psi_1 \cdots f_n \doteq \Psi_n]$$

where X is a variable in $\mathcal{V}$, s is a sort in $\mathcal{S}$, $f_1, \cdots, f_n$ are features in $\mathcal{F}$, $n \geq 0$, and each Ψ_i is either a feature term or a set of feature terms. We also identify a feature term with the singleton set of that feature term.

Note that when $n = 0$ we are defining only a sorted variable $(X : s)$. We call the variable X in the above feature term the *root* of ψ, and say that X is *sorted* by the sort s (noted $Sort(\psi)$) and has features $f_1, \cdots, f_n$.

Using this syntax for feature terms, the following expression (named ψ_1)

$$\psi_1 = X : Person \left[\begin{array}{ll} last\_name \doteq \text{Smith} & \\ drives \quad \doteq Y : Car \left[\begin{array}{l} owner \doteq X \\ model \doteq Z : Ibiza \end{array} \right] & \end{array} \right]$$

is an example of a feature term denoting persons whose last_name is Smith, who drive an Ibiza-model car of which she/he is also the owner.

A feature term is a syntactic expression that denotes sets of elements in some appropriate domain of interpretation ($[\![\psi]\!]^{\mathcal{I}} \subset \mathcal{D}^{\mathcal{I}}$). Thus, given the previously defined interpretation $\mathcal{I}$, the denotation $[\![\psi]\!]^{\mathcal{I}}$ of a feature term ψ, under a valuation $\alpha : \mathcal{V} \mapsto \mathcal{D}^{\mathcal{I}}$ is given inductively by:

$$[\![\psi]\!]^{\mathcal{I}} = [\![X : s[f_1 \doteq \psi_1 \cdots f_n \doteq \psi_n]]\!]^{\mathcal{I}} = \{\alpha(X)\} \cap s^{\mathcal{I}} \bigcap_{1 \leq i \leq n} (\ell_i^{\mathcal{I}})^{-1}([\![\psi_i]\!]^{\mathcal{I}})$$

where $F^{-1}(S)$, when F is a function and S is a set, stands for $\{x | \exists S' \supset S$ such that $F(x) = S'\}$; i.e., denotes the set of all elements whose images by F contains at least S.

Using this semantical interpretation of feature terms, it is legitimate to establish a relation order between terms. Given two terms ψ and ψ', we will be interested in determine when $[\![\psi]\!]^{\mathcal{I}} \subset [\![\psi']\!]^{\mathcal{I}}$.

2.2 Subsumption

The semantical interpretation of feature terms brings an ordering relation among feature terms (also called *descriptions* in Noos). We call this ordering relation *subsumption*. The intuitive meaning of subsumption is that of *informational ordering*. We say that a feature term ψ subsumes another feature term ψ' ($\psi \sqsubseteq \psi'$) when all information in ψ is also contained in ψ'—i. e. corresponds to the semantical relation of $[\![\psi']\!]^{\mathcal{I}} \subset [\![\psi]\!]^{\mathcal{I}}$.

Definition 2. (Subsumption)
Given two feature terms ψ and ψ', ψ subsumes ψ', $\psi \sqsubseteq \psi'$, when :

1. $Sort(\psi) \leq Sort(\psi')$, and
2. for every $f_i \doteq \Psi_i$ defined in ψ then f_i has to be defined in ψ' as $f_i \doteq \Psi_i'$ and exists a total injective function $h : \Psi_i \rightarrow \Psi_i'$ such that $\forall \psi_k \in \Psi_i \ \psi_k \sqsubseteq h(\psi_k)$.

For instance, consider the previous presented example of a feature term (ψ_1) and the following one (ψ_2) denoting persons driving a car with an owner and any car model:

$$\psi_2 = X : Person \left[drives \doteq Y : Car \left[\begin{array}{l} owner \doteq V : Person \\ model \doteq Z : Car\_model \end{array} \right] \right]$$

Clearly $\psi_2 \sqsubseteq \psi_1$, i. e. this term subsumes the previous one. Notice here that `owner` in ψ_2 has variable V different from X—an equality that was enforced in ψ_1 since both had the same variable X. In other words, ψ_1 has more constraints (more information) than ψ_2 but also all information in ψ_2—thus the subsumption holds.

Finally, we introduce the notion of *equivalence* among feature terms:

Definition 3. (Equivalence) Given two feature terms ψ and ψ', we say that they are syntactic variants if and only if $\psi \sqsubseteq \psi'$ and $\psi' \sqsubseteq \psi$.

Anti-unification (AU) in feature terms is defined in the classical way (as the "least common subsumer" or "most specific generalization") over the subsumption lattice as follows.

Definition 4. (Anti-unification) The anti-unification of two terms $\psi \sqcap \psi'$ is an upper lower bound with respect to the subsumption ($\sqsubseteq$) ordering.

Notice that the upper lower bound of two terms need not be unique—i. e. there may be several terms that are an upper lower bound and are not equivalent. An algorithmic definition of AU is given in § 3. Moreover AU can also be used in Case-based Reasoning for establishing a symbolic representation of similitude and reasoning about similitudes [11].

2.3 Understanding Feature Terms as Clauses

There a several syntaxes amenable to represent feature terms. We have used up to now a record-like syntax, but graphs and clausal syntaxes can also be used. Using labeled graphs (1) arcs are labeled with predicates symbols, (2) nodes stand for sorted variables where the sort symbol is the node label, and (3) path[1] equality is tantamount to variable symbol equality. An example is the induced description of a class of marine sponges as shown at Figure 5 (notice that feature *size* has a set value with two elements *Megas* and *Micros* but no path equality appears).

Feature terms can be also understood as conjunctions of clauses[2]. This clausal representation is useful and more usual for ML methods. There are two kinds of atomic clauses: sort clauses ($X : s$) and feature clauses ($f(X, Y)$). A given feature term can be represented also as a conjunction of these two kind of atomic clauses. Thus, for a term $\psi = X : s[f_1 \doteq \psi_1 \cdots f_n \doteq \psi_n]$, a clausal form $\phi(\psi)$ is built as follows $\phi(\psi) = X : s \wedge f_1(X, Y_1) \wedge \phi(\psi_1) \wedge \cdots \wedge f_n(X, Y_n) \wedge \phi(\psi_n)$, where $Y_1, \cdots, Y_n$ are roots of $\psi_1, \cdots, \psi_n$ respectively.

For instance, the first example of feature term (ψ_1) is represented in clausal form as follows:

$$\begin{aligned} X : Person \wedge \quad & lastname(X, \text{Smith}) \quad \wedge \quad drives(X, Y) \\ \wedge \quad & Y : Car \quad \wedge \quad owner(Y, X) \quad \wedge \quad model(Y, Z) \ \wedge \ Z : Ibiza \end{aligned}$$

2.4 The Inductive Setting

The process of induction over feature terms with the goal of finding a *discriminant description* can be specified as follows:

Given

1. a set of positive E^+ and negative E^- examples.
2. a notion of *subsumption*
3. Background knowledge B in the form of domain methods

[1] A path is a sequence of arcs.

[2] The reader interested in the precise mapping among the different representations (and the semantic consistency of the subsumption lattice across these mappings) is referred to [1].

Find a feature term (description) ψ such that $\forall\, e^+ \in E^+ : \psi \sqsubseteq e^+$ and $\forall\, e^- \in E^- : \psi \not\sqsubseteq e^-$.

Notice that AU of positive examples provides a way for achieving the first condition $e^+_{AU} = e^+_1 \sqcap e^+_2 \ldots \sqcap e^+_n \Rightarrow e^+_{AU} \sqsubseteq e^+ : \forall\, e^+ \in E^+$, namely the least general generalization of E^+ subsumes all positive examples but it may also subsume some negative example. Next section describes the heuristic strategy used by INDIE for exploiting the AU operation to search for a discriminant description in the subsumption lattice of feature terms.

In the inductive setting presented here we have included a given (3), namely background knowledge B, that will be now summarily explained. Background knowledge is expressed in Noos by means of methods that infer the values of features. A feature term is *closed* when all methods had inferred values for the features. The AU of two feature terms requires them to be closed, so the Noos AU method forces the domain methods to be applied in a lazy on-demand way. Method definition has not been included in the syntax introduced in this section for reasons of space and because it is not needed for the domains shown in §4 (but see [2]). A recension of feature term induction with background knowledge is to wait for another article.

3 The Inductive Method INDIE

INDIE is a heuristic bottom-up inductive learning method that obtains a most specific generalization subsuming a set of positive examples. The main contribution of INDIE is handling objects represented as feature terms. INDIE can work either on positive examples only or on positive and negative examples. Working on positive examples only requires AU and it allows to solve the characterization task (also called discovery or description problem), useful in KDD and Data-Mining. Working on positive and negative examples allows to perform the discrimination task (also called prediction problem or concept learning) that is the usual in most ML applications. In the following we describe INDIE in the discrimination task.

3.1 Description of the Method

Given a set of training examples $E = \{e_1, ..., e_m\}$ and a set of solution classes $C = \{C_1, ..., C_n\}$, the goal of INDIE is to obtain a discriminant description D_k for each solution class C_k. Each example e_i is a feature term having a subset of features $A_i = \{A_{i1}, ..., A_{ik} | A_{ij} \in \mathcal{F}\}$. Training examples can have a different subset of legal features in $\mathcal{F}$. Each feature A_i has as values a set of objects O_i where each $o_{ij} \in O_i$ is a feature term that can have, in turn, a set of features.

A description $D_k = \{d_{jk}\}$ represents a disjunction of feature term descriptions for the current solution class C_k. Each d_{jk} subsumes a subset of positive examples of C_k and does not subsume negative examples. In a discriminant task, negative examples of a solution class C_k are all those training examples that do not belong to C_k.

```
Function INDIE (E+, E-)
    D = ∅
    D_k = Anti-unification (E+) ; most specific generalization
    if D_k ⊑ e for some e ∈ E-
        then A_l = {A_i| features in D_k chosen according to leaf bias}
             P_N = Discriminant-partition (A_l,E+,E-)
             for each set S_i ∈ P_N do
                 D_i = INDIE(S_i, E-)
                 Add D_i to D
             end-for
        else Add D_k to D
    end-if
    Eliminate any d ∈ D such that d ⊑ d' ∈ D
    return D
end-function
```

Fig. 1. INDIE obtains a disjunction of descriptions D_k that do not subsume negative examples for the current class.

Given a set of positive examples E^+ for a solution class C_k the INDIE algorithm (Fig. 1) obtains, using the AU operation, a most specific generalization D_k subsuming all the examples in E^+. If the description D_k does not subsume negative examples then D_k is a correct description for C_k. Because the value of a feature can be a set, several most specific generalizations subsuming all the positive examples can be built. This means that if one of the most specific generalizations D_k subsumes some negative example there is two options to solve this situation: 1) to search for another most specific generalization D_j that does not subsume negative examples, or 2) to specialize D_k until no negative examples are subsumed. Using the first option all the possible most specific generalizations are tested searching for a description D_k that does not subsume negative examples. If all the descriptions D_k subsume negative examples, the second option has to be taken. The second option assumes that the only way to describe the current solution class C_k is using a disjunction of descriptions.

The next question is, how many descriptions are necessary to describe C_k? To answer this question a heuristic approach is taken. INDIE selects the most relevant feature A_d (using the *discriminant − partition* function explained later) that generates a partition P_N of E^+ in N classes, where N is the number of different values that A_d takes in E^+. Thus, if D_k subsumes some negative examples there is at most a disjunct of N descriptions for C_k, i.e., $D_k = \{d_{jk}\}$ for $j = 1$ to N. The new specialized description is a disjunction $D_k = \{d_{jk}\}$ recursively obtained by applying the INDIE algorithm to each set of the partition P_N. This process is repeated until D_k does not subsume negative examples or all the features have been used. If there are two descriptions d_{1k} and d_{2k} in D_k such that $d_{1k} \sqsubseteq d_{2k}$

```
Function AU2 (E1, E2, D)
    Let D be a new term with Sort(D) = Sort(E1) ⊓ Sort(E2)
    A = {Ai| common attributes to E1 and E2}
    for each Ai ∈ A do
        Vi = (vi1, vi2) where vi1 = E1.Ai and vi2 = E2.Ai
        if vi1 = vi2
            then add-feature (D, Ai, vi1)
            else if there is a p ∈ *paths* such that p = (Vi, di)
                    then add-feature (D, Ai, di)
                    else Let di be a new term with Sort(di) = Sort(vi1) ⊓ Sort(vi2)
                        add (Vi, di) to *paths*
                        if vi1 and vi2 have zero features
                            then add-feature (D, Ai, di)
                            else add-feature (D, Ai, AU2 (vi1, vi2))
        endif endif endif
    end for
end function
```

Fig. 2. Anti-unification operation that constructs the most specific generalization covering a given set of positive examples. $Add - feature(d, a, v)$ is a function that adds the feature a with value v to the description d.

then d_{2k} can be eliminated and the disjunct is simplified.

In the next sections, we explain the main steps of INDIE, i.e. how to construct a most specific generalization (AU operation), which bias is used to select the set of features that allow the to define a partition over the positive examples and how the most discriminant partition is selected (discriminant-partition operation).

3.2 Anti-unification

The goal of the anti-unification (AU) operation is to construct a most specific generalization D from a set of positive examples E^+ of the current solution class C_k. Figure 2 shows the algorithm AU2 used to obtain a most specific generalization of two examples E_1 and E_2. The first step is to obtain the set of features that are common to both E_1 and E_2. We will explain AU2 assuming the attributes have only one value, and later we will explain the case when the values of an attribute are sets. For each common attribute A_i, let us consider the pair $V_i = (v_{i1}, v_{i2})$, where $v_{i1} = E_1.A_i$ (the value taken by the A_i in the example E_1) and $v_{i2} = E_2.A_i$. Each pair V_i has associated an object d_i that is the description obtained from the AU of the values contained in V_i. All the pairs (V_i, d_i) are stored in the *paths* variable (see Figure 2) in order to detect if a particular combination of values has already been anti-unified. Whenever the algorithm founds a pair (v_{i1}, v_{i2}) already contained in *paths*, its associated object d_i is the value for $D.A_i$. This process assures path equality. When the

pair $V_i = (v_{i1}, v_{i2})$ has not previously appeared, the values v_{i1} and v_{i2} have to be anti-unified.

Feature terms in Noos, as we saw, are set-valued. Let us suppose that $S_{i1} = E_1.A_i$ and $S_{i2} = E_2.A_i$ are sets of values. The AU of S_1 and S_2 has to find a set of values S such that: 1) $Card(S) = min\{Card(S_{i1}), Card(S_{i2})\}$, and 2) each $s_i \in S$ is obtained from the AU of two different values $v_j \in S_1$ and $v_k \in S_2$. The AU2 algorithm is applied to each possible pair (v_j, v_k) where $v_j \in S_1$ and $v_k \in S_2$, obtaining a set $\{g_p\}$ containing $Card(S_1) \times Card(S_2)$ descriptions. From this set, a most specific combination[3] of $Card(S)$ elements has to be taken as the value set of the attribute A_i. Note that can exist several incomparable combinations that are maximally specific. AU2 randomly chooses one of them. The AU of n examples $E_1, ..., E_n$ consists of applying the AU2 algorithm n-1 times, starting by computing $D_1 = \text{AU2}(E_1, E_2)$ and iterating $\text{AU2}(D_{i-2}, E_i)$ over $i = 3 \ldots n$.

3.3 Bias and Partitioning the Set of Positive Examples

The discriminant-partition function (see Figure 3) determines the most discriminant feature A_d (among those features belonging to $\mathcal{F}$).We consider as candidates to be the most discriminant feature only those features appearing in D_k. This bias reduces the set of candidates to those features appearing in all the positive examples. In a structured representation two kinds of features can be distinguished: leaf features and intermediate features (those belonging to intermediate levels of the structured representation). For example, in the sponge shown in Figure 4, some leaf features are *axis* or *grow* and some intermediate features are *size* or *micros*. Usually, existing inductive learning methods select one predicate at time to specialize a clause. If we select an intermediate feature as the most discriminant, its value is a subterm and this means that the description D_k is specialized according the type of that subterm. This situation is equivalent to specialize a clause introducing several predicates at time. The selection of only one predicate is achieved in feature terms by selecting a leaf feature. Therefore, our bias is to consider as candidates to be the most discriminant feature the set of leaf features of D_k. In practice, the selection of an intermediate feature to specialize the description D_k tends to produce descriptions too specific (in the sense that it quickly reaches a disjunction of M descriptions, where M is the number of positive examples). In principle, we are interested in a description of a solution class using the least number of descriptions.

To select the most discriminant feature we take the minimum distance between partitions—using the López de Mántaras distance [8]. Given two partitions P_A and P_B of a set S, the distance among them is computed as follows:

$$d_N(P_A, P_B) = 2 - \frac{I(P_A) + I(P_B)}{I(P_A \cap P_B)}$$

[3] A most specific combination is such that is not subsumed (in the sense of definition 2) by any other combination.

```
Function DISCRIMINANT-PARTITION (A_l, E+, E-)
    Dist = ∅
    while A_l ≠ ∅ do
        P_c = ((E+)(E-)) ;; the correct partition
        for A_i ∈ A_l do
            P_i = {S_i ⊂ E |∀v_i ∈ S_i and ∀v_j ∈ S_j : Sort(v_i) ≠ Sort(v_j)}
opez de M   D_i = D(P_c, P_i) ;; L
            Add D_i to Dist
        end-for
    end-while
    useful-attribute = false
    while Dist ≠ ∅ and (useful-attribute = false) do
        d_min = min{D_i ∈ Dist}
        Let A_min and P_min the attribute and the partition associated to d_min
        P_d = {S'_i|S'_i = S_i - E- : S_i ∈ P_min}
        if P_d has only one non-empty S'_i
            then Remove d_min from Dist
            else useful-attribute = true
        end-if
    end-while
    if Dist = ∅ then return E+
                else return P_d
    end-if
end-function
```

Fig. 3. Discriminant-partition function selects the most useful attribute in a description leaf using the López de Mántaras distance. The set of relevant attributes is given in A_l by algorithm in Fig. 1

where $I(P)$ is the information of a partition P and $I(P_A \cap P_B)$ is the mutual information of two partitions.

In our case, the distance measure is applied to compute the distance among a partition generated by a feature and the correct partition. The correct partition P_c has two classes, one containing the positive examples (examples in C_k) and the other containing the negative examples (those not in C_k). Thus, for each feature $A_i \in A_l$ in a description leaf of the current description, the set of training examples $\mathrm{E} = E^+ \cup E^-$ is partitioned according the sorts of the values of A_i generating a partition P_i. Each partition P_i is compared with the correct partition P_c using the López de Mántaras distance. The most discriminant feature A_d is that producing a partition P_d having the minimum distance $D_d(P_d, P_c)$ to the correct partition P_c.

Generalization Post-process After applying INDIE, an optional post-processing step can be used. Since D_k is a most specific generalization for a

solution class C_k, it can be generalized (in principle) without subsuming any negative example. The post-process consists of eliminating features as far as no negative examples are subsumed For each description $d_{jk} \in D_k$, the algorithm for post-processing uses the López de Mántaras distance to rank all the features belonging to d_{jk}. The features are considered from the least discriminant to the most discriminant.

4 Experiments with INDIE

In this section we show the results of applying INDIE to several domains: robots, drugs, marine sponges, and lymphographies. We have chosen these datasets to show and evaluate different aspects of this feature term induction method. In the following subsections the results of these experiments are explained.

4.1 Concept Learning using INDIE

The robots domain [10] consists of a description of six robots that belongs to two solution classes: *friendly* and *unfriendly*. The robots are described using an attribute-value representation. However, using the feature term formalism obtains a relational definition for the *friendly* class:

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

i. e. the robots that have the same shape of body and of head belong to the *friendly* class.

The drugs domain consists in a description of several drugs and is used by the KLUSTER system [9] . From these descriptions KLUSTER can obtain several classifications, i.e. what is an active substance, what is a monodrug, when a substance is sedative, etc. To represent the domain objects KLUSTER uses a representation language based on KL-ONE. In INDIE we have represented the domain objects as feature terms and the goal is to obtain a description for the solution classes *monodrug*, *combidrug* and *placebo*. The obtained descriptions for these classes are similar to those obtained by the KLUSTER system. The following are descriptions obtained by INDIE for the *monodrug*, *combidrug* classes.

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

$$\text{Combidrug} = X : drug \left[contains \doteq \begin{matrix} Y : active\_substance \\ Z : active\_substance \end{matrix} \right]$$

The differences are due to the different representation (see §5 for a comparison of it with INDIE). Notice that the main difference between both classes is using one active substance (*monodrug*) and more than one active substance (*combidrug*). This fact derives from the definition of subsumption, namely that any example

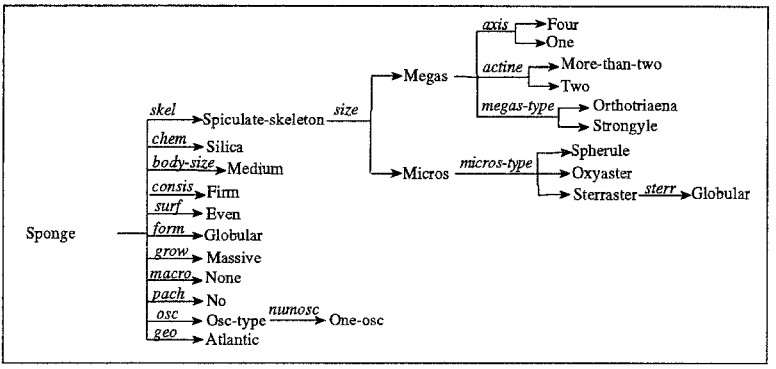

Fig. 4. A *graph* description of a feature term for a specimen of marine sponge (of the *Erylus discophorus* species).

subsumed by the *Combidrug* description above, needs to have (at least) two active substances for feature *contains*. These examples would also be subsumed by *Monodrug* description (they also have *one* active substance) except that they do not satisfy the other two features (*effects* and *affects*) .

We have employed INDIE in the multistrategy learning system for marine sponge identification SPIN. Marine sponges offer a good test bed for the capability of feature terms for dealing with incomplete descriptions, since marine sponges variability involves that certain properties are relevant only for certain types and/or certain subcomponents of them. Given a set of sponges correctly classified INDIE has been capable to obtain a description for the different taxa. Sponges are represented as feature terms. See an example of a marine sponge in Figure 4 using the graph syntax for feature terms. Marine sponges can be described by a great variety of features and they form a domain where partial descriptions ("incomplete information") are commonplace. For example, a marine sponge of the *Erylus discophorus* species may be described by the skeleton, the geographic localization, the form and the color whereas other sponge only may be described by the skeleton. Figure 5 shows the descriptions obtained by INDIE for the *Erylus* taxon.

Also of interest are the results of INDIE in the family relations domain. Notice that INDIE infers descriptions (e. g. of persons that are uncles or mothers) and not rules to compute the relations uncle or mother. For this concepts, INDIE finds a correct and discriminating description with AU only, and the discriminating stage is not used. In this situation, INDIE's anti-unification (AU) stage is closer to the characterization task—discovering regularities as found in data mining or in the so-called description problem—indeed AU finds by definition *all* those common to a set of (positive) examples. Let us consider those persons that are mothers and those that are uncles in the family relations dataset. In the both cases INDIE finds using AU a description that subsumes only positive examples

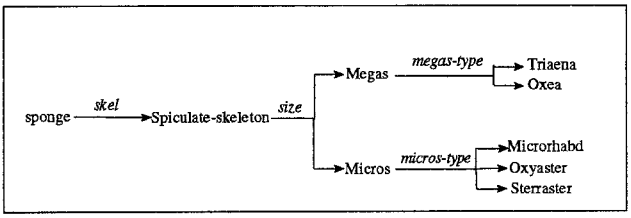

Fig. 5. A *graph* description of a feature term for the description of the *Erylus* genus).

and no negative examples (those persons that are not, respectively, mothers or uncles). In the case of the mother description, AU finds the following term:

$$X : female \left[\begin{array}{l} son \doteq Y : male \left[\begin{array}{l} father \doteq W \\ mother \doteq X \\ sister \doteq Z \end{array} \right] \\ daughter \doteq Z : female \left[\begin{array}{l} father \doteq W \\ mother \doteq X \\ brother \doteq Y \end{array} \right] \\ husband \doteq W : male \left[\begin{array}{l} son \doteq Y \\ daughter \doteq Z \\ wife \doteq Y \end{array} \right] \end{array} \right]$$

All this regularities hold for all the involved persons in the dataset. After the optional generalization postprocess only one of the regularities remains, namely

$$X : female \left[husband \doteq W : male \left[son \doteq Y : male \right] \right]$$

that clearly is correct—but other regularities that have very similar heuristic values have been eliminated, so they could have equally been chosen (like having a son or a daughter).

The uncles description also induces a number of regularities by AU, for instance, the niece X has the following relation with the uncle Z:

$$X : female \left[mother \doteq Y : female \left[brother \doteq Z : male \right] \right]$$

and also the following one, expressed in clausal form:

$$\begin{array}{llllllll} X : female & \wedge & father(X,Y) & \wedge & Y : male & \wedge & sister(Y,W) & \wedge \\ W : female & \wedge & husband(W,Z) & \wedge & Z : male & \wedge & uncle(X,Z) & . \end{array}$$

While ILP systems are more biased to single predicate learning, this example shows that feature term induction is biased toward "multiple predicate learning"—in the example above, a description of niece has been found in relationship to the uncle description. After the simplification postprocess the only following regularity remains:

$$X : male \left[wife \doteq Y : female \left[niece \doteq Z \right] \right]$$

4.2 The Accuracy of INDIE

We have evaluated the accuracy of INDIE in order to assess the utility for INDIE of using the López de Mántaras heuristic with respect to attribute-value learners of the decision tree family. The accuracy has been evaluated performing some experiments under the same conditions that those described in [12], i.e. the 67% of the cases have been taken for learning and the remaining 33% have been taken for test. We will now summarize the results on the lymphographies data set of the Irvine ML Repository. The accuracy for C4.5, CN2 and PIK are respectively 76.4, 81.7/76.5 (corresponding to two parameter setting), and 77.2.

A main difference of INDIE is that it can provide multiple solutions, situation that may be useful in some domains (i.e. for the sponge classification) or not. If we consider that multiple solutions including the correct one is a correct solution then the accuracy of INDIE is 81.4. However if we consider that a multiple solution is incorrect even if it contains the correct solution then the accuracy of INDIE is 76.63.

5 Discussion

There has been relatively few work on induction on "structured representations" compared to the intensive research performed on Horn clause representations. The KLUSTER system [9] and the LCSLearn algorithm [6] work on some description logics (a KL-One-like language). As we said of description logics, these formalisms are related to but different from feature terms. A main semantic difference is that feature terms provide a *uniform* representation while description logics are *hybrid*—there are two different formalisms, one for the describing concepts (T-box) and another one for describing instances (A-box).

KLUSTER searches for a most specific generalization MSG from positive examples. If MSG covers negative examples KLUSTER follows a particular algorithm to specialize the MSG by means of introducing new `at-most` and `at-least` predicates in the feature descriptions. INDIE uses AU to find a most specific description that subsumes positive examples (similarly to KLUSTER since both follow a bottom-up strategy) and then to specialize the description INDIE introduces disjunction of descriptions following a distance-based heuristic. KLUSTER has been applied only to the drugs domain we showed in §4.

The LCSLearn algorithm is a bottom up inductive method for C-Classic, a subset of the Classic description logic language that has all Classic constructors except for the `same-as` constructor, that is roughly equivalent to *path equality* in feature logics. Since feature logics and INDIE depend heavily on the notion of path equality, both LCSLearn and INDIE seem complementary in studying formalisms that embody different useful subsets of first order logic. However, the LCSLearn is comparable only to the AU step in INDIE —the LCSLearnDisj

algorithm, the disjunctive induction version of LCSLearn, is more akin to INDIE in that it is able to induce a disjunction of description. While LCSLearn is shown to be pac-learnable, nothing is said in [6] about LCSLearnDisj. While INDIE uses a distance-based heuristic to select how to split a description into a disjunction of descriptions, LCSLearnDisj essentially chooses positive and negative examples in a random way until a disjunctive discriminant description is reached.

Subsumption is a natural way to relate to induction. Classical ML methods did so while ILP methods have to deal with deductive relations—from which a generalization relation has to be defined. Although the relation of the expressiveness of feature term formalisms and Horn clauses formalism is an open issue in the current research literature, it is possible that the first can be a subset of the latter. If that would be the case, the less expressive formalism of feature terms—closer to object-oriented representations—could be more appropriate for application domains that do not require the full induction of logic programs since they can work on the well defined subsumption lattice of feature terms.

The formalization introduced here has to be enriched to include domain methods in Noos performing inference. INDIE is already applicable since it requires *closed* feature terms and the AU operation forces method evaluation generating the closed feature terms. Future experimental work on several application domain is needed to show learning of feature terms with background knowledge.

Incomplete information has a natural representation in feature terms that base the subsumption relation in the notion of information ordering. A subsumed description is a *refined* term—a description with more refined (more specific sorts) or additional (new features) information.

The explanation of the framework for combining multiple learning methods in SPIN is left out for lack of space but the reader is referred to [3] for such an explanation regarding the multistrategy learning system CHROMA. The SPIN system currently combines INDIE with CRASS, a lazy learning method that uses an entropy-based measure to estimate the better example to perform an sponge identification by case-based reasoning [11].

Future work includes finishing a formal study of subsumption and AU complexity. Currently we know complexity stems from set-valued features —in fact, subsumption in feature terms without set values is lineal with the number of nodes and features [5]. We know that subsumption is much more costly only when we have embedded set-values. That is, if $D_1.A_i = S$ where S is a set, and there is a term $D_2 \in S$ that also has a set valued attribute, say $D_2.A_j = S'$, then we say S' is an embedded set-value at depth one. We estimate the complexity of dealing with embedded set-values in subsumption to be $O(n^k)$ where n is the maximum cardinality of a set and k is the embedding depth. In practice this means that while representing some application domain a user can have a clear idea of the expected complexity just looking at the level of embedding.

Related to complexity two aspects of INDIE are currently being developed. One is the effect of using a weaker form of subsumption—roughly, one that does not require in set subsumption that the subsumed elements are different. This weaker form decreases the language expressiveness, since the constraint on a

minimum set cardinality (as used in the drugs domain) is dropped, but may also decrease complexity. A second aspect is parameterizing INDIE with a maximum node number to be considered during AU—equivalent to a maximum number of variables to be considered. Other options are also to limit the depth or number of features; the relationship of this bias to ij-determinacy in Horn clauses induction is also interesting. Lastly, we are developing a second inductive method using a declarative bias mechanism constraining the form of feature terms to be searched during induction.

Acknowledgments The authors thank Josep-Lluís Arcos and Ramon López de Mántaras for the discussions and the support he provided in this and other closely related work. The research reported on this paper has been developed at the IIIA inside the SMASH Project funded by Spanish CICYT grant TIC-96-1038, and a CICYT fellowship. Information about this and updated results will be posted at `http://www.iiia.csic.es/Projects/learning.html`.

References

1. H. Aït-Kaci and A. Podelski. Towards a meaning of LIFE. *J. Logic Programming*, 16:195–234, 1993.
2. J. L. Arcos and E. Plaza. Inference and reflection in the object-centered representation language Noos. *Journal of Future Generation Computer Systems*, 12:173–188, 1996.
3. E. Armengol and E. Plaza. Integrating induction in a case-based reasoner. In J. P. Haton, M. Keane, and M. Manago, editors, *Advances in Case-Based Reasoning*, number 984 in Lecture Notes in Artificial Intelligence, pages 3–17. Springer-Verlag, 1994.
4. F. Bergadano and D. Gunetti. *Inductive Logic Programming. From Machine Learning to Software Engineering.* The MIT Press, 1995.
5. B. Carpenter. *The Logic of Typed Feature Structures.* Tracts in theoretical Computer Science. Cambridge University Press, Cambridge, UK, 1992.
6. W. W. Cohen and H. Hirsh. Learning the classic description logic: Theoretical and experimental results. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference*, 1994.
7. L. Dami. *Software Composition: Towards an Integration of Functional and Object-Oriented Approaches.* PhD thesis, University of Geneva, 1994.
8. R. López de Mántaras. A distance-based attribute selection measure for decision tree induction. *Machine Learning*, 6:81–92, 1991.
9. J.-U. Kietz and K. Morik. Polynomial induction of structural knowledge. *Machine Learning*, 14:193–217, 1994.
10. N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Tecniques and applications.* Ellis Horwood, 1940.
11. E. Plaza, R. López de Mántaras, and E. Armengol. On the importance of similitude: An entropy-based assessment. In Boi Faltings et al, editor, *Case-Based Reasoning, EWCBR-96*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1996.
12. X. M. Zhou and T. S. Dillon. Theoretical and practical considerations of uncertainty and complexity in automated knowledge acquisition. *IEEE Trans. Knowledge and Data Engineering*, 7:699–712, 1995.