Refinement-based Disintegration: An Approach to Re-representation in Relational Learning

Santiago Ontañón [a] and Enric Plaza [b]
[a] *Computer Science, Drexel University Philadelphia 19104, PA, USA, e-mail: santi@cs.drexel.edu*
[b] *IIIA, Artificial Intelligence Research Institute CSIC, Spanish Council for Scientific Research Campus UAB, 08193 Bellaterra, Catalonia (Spain), e-mail: enric@iiia.csic.es*

We present a new approach lo learn from relational data based on re-representation of the examples. This approach, called *property-based re-representation* is based on a new analysis of the structure of refinement graphs used in ILP and relational learning in general. This analysis allows the characterization of relational examples by a set of multi-relational patterns called *properties*. Using them, we perform a property-based re-representation of relational examples that facilitates the development of relational learning techniques. Additionally, we show the usefulness of re-representation with a collection of experiments in the context of nearest neighbor classification.

Keywords: Relational Learning, Re-representation, Refinement Operators, Feature Terms, Propositionalization

1. Introduction

Relational Machine Learning (RLM) studies how to design machine learning algorithms for domains where data is structured, or relational. Moreover, conventional machine learning approaches [15] assume data is represented in a propositional (feature-vector) approach. Two approaches are commonly used to address this problem: the definition of machine learning approaches that can deal directly with relational representations, and *propositionalization* approaches [11] that translate relational data to a feature-vector representation where conventional machine learning approaches can be used.

In this paper we present a new approach lo learn from relational data, similar to propositionalization, based on re-representation of the examples. While propositionalization convert relational data to a vector of propositional features, our new approach converts relational data to a collection of *properties*, that can be shown not to lose any information under certain circumstances. This approach, called *property-based re-representation* is based on a new analysis of the structure of refinement graphs used in Inductive Logic Programming (ILP) and relational learning in general. This analysis allows the characterization of relational examples by a set of multi-relational patterns called *properties*, with which we perform a property-based re-representation of relational examples that facilitates the development of relational learning techniques.

Our approach is based on the notion of *refinement graphs* [12], which are commonly used in ILP [13] and other areas, such as Description Logics, for performing inductive relational learning [4,19,20]. A refinement graph in the space of generalizations is built by defining a suitable *refinement operator*, which can specialize a given description to form more specific descriptions. Given the most general description that can be expressed in a given representation language, refinement operators can generate the whole space of expressions in such language by iteratively refining (i.e. specializing or generalizing) such description.

The approach and techniques presented here can be used to apply any propositional machine learning algorithm to relational data, and are, in principle, applicable to any representation language for which we can define a refinement operator that satisfies certain properties, namely they have to be complete and locally finite. For reasons of space an clarity, however, this paper will focus on a particular representation formalism, namely *feature terms* (presented in the next section). Then, we present the notion of a *property* (a multi-relational pattern) of a relational example, and the *dis-*

integration operation, which splits a given relational example in a collection of properties. These properties can later be integrated again if need be to reconstruct the original example. Moreover, we introduce the way to re-represent an example as a set of properties and the building of a vocabulary of properties to represent examples in a given data set. The reason to present this re-representation is that it allows to use classical propositional machine learning techniques (with a small adaptation) to relational data, as discussed in Section 8. After showing the usefulness of our approach with an empirical evaluation in the context of nearest-neighbor classification, we discuss the relation of our approach with existing approaches such as propositionalization. The paper closes with related work and conclusions.

2. Preliminaries

This section introduces the formalism of feature terms, and the basic notions of refinement operators, over which we based the work presented in this paper.

2.1. Feature Terms

Feature terms [2,6] (also called typed feature structures, or Ψ-terms) are a generalization of first-order terms, introduced in theoretical computer science to formalize object-oriented declarative languages. Feature terms correspond to a different subset of first-order logics than Description Logics. However, they have the same expressive power —only differing in their basic reasoning mechanisms [1].

Feature terms are defined by its *signature*: $\Sigma = \langle \mathcal{S}, \mathcal{F}, \leq, \mathcal{V} \rangle$. $\mathcal{S}$ is a set of sort symbols, including $\perp$ representing the most general sort ("any"), and $\top$ representing the most specific sort ("none"); $\leq$ is an order relation inducing a single inheritance hierarchy in $\mathcal{S}$, and where $\perp \leq s \leq \top$ for each sort $s \in \mathcal{S}$. Specifically, $s \leq s'$ means s is more general than or equal to s', for any $s, s' \in \mathcal{S}$ ("any" is more general than any s which, in turn, is more general than "none"). $\mathcal{F}$ is a set of feature symbols, and $\mathcal{V}$ is a set of variable names. We define a feature term ψ as,

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

where ψ points to the *root* variable X (that we will note as $root(\psi)$) of sort s; $X \in \mathcal{V}$, $s \in \mathcal{S}$, $f_i \in \mathcal{F}$, and Ψ_i is a set of variables $\{X_1, ..., X_m\}$ (when the size of this set is larger than 1, we say that there is a *set-valued feature*).

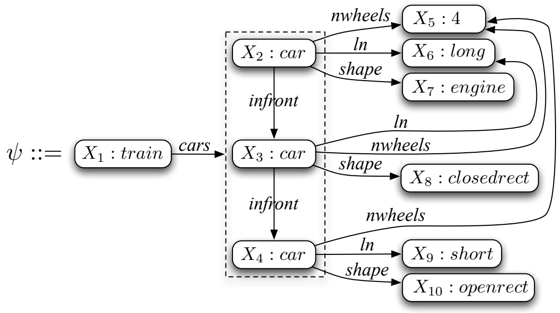


Fig. 1. A train represented as a feature term.

When the value of some feature of two variables $X_i.f_1$ and $X_j.f_2$ share some variable, i.e. $X_i.f_1 \cap X_j.f_2 \neq \emptyset$, we say that there is a *variable equality*.

An example feature term appears in Figure 1. It is a train (variable X_1) composed of three cars (variables X_2, X_3 and X_4). This term has 10 variables, and one set-valued feature (its values enclosed in a dashed line): There are also several *variable equalities*, e.g. equality $X_3.infront = X_4$. means that the car in front of car X_3 is car X_4. The set of variables of a term ψ is $vars(\psi)$, the set of features of a variable X is *features(X)*, and *sort(X)* is its sort. The basic relation over feature terms is *subsumption* ($\sqsubseteq$), i.e. whether a term is more general (or equal) than another[1].

Definition 1. *(Subsumption) A feature term ψ_1 subsumes another one ψ_2 ($\psi_1 \sqsubseteq \psi_2$) when there is a total mapping m: $vars(\psi_1) \rightarrow vars(\psi_2)$ such that:*

1. *$root(\psi_2) = m(root(\psi_1))$, and*
2. *$\forall X \in vars(\psi_1)$*
   (a) *$sort(X) \leq sort(m(X))$, and*
   (b) *$\forall f \in features(X)$, where $X.f = \Psi_1$ and $m(X).f = \Psi_2$, we have that:*
      i. *$\forall Y \in \Psi_1, \exists Z \in \Psi_2 : m(Y) = Z$,*
      ii. *$\forall Y, Z \in \Psi_1, Y \neq Z \Rightarrow m(Y) \neq m(Z)$*

      i.e. each variable in the set Ψ_1 is mapped to a variable in Ψ_2, and each different variable in Ψ_1 has a different mapping.

If $\psi_1 \sqsubseteq \psi_2$ and $\psi_2 \sqsubseteq \psi_1$, we say that they are *equivalent*: $\psi_1 \equiv \psi_2$. Subsumption induces a partial order over the set of all feature terms, i.e. the pair $\langle \mathcal{L}, \sqsubseteq \rangle$ is a *poset*, where $\mathcal{L}$ is the set of all feature terms that can be

[1]In description logics notation, subsumption is written in the reverse order since it is seen as "set inclusion" of their interpretations. In ML terms, $A \sqsubseteq B$ means that A is more general than B, while in description logics it has the opposite meaning.

formed given a signature, and that contain the infimum $\perp$ and the supremum $\top$ with respect to the subsumption order. The subsumption relation allows us to view the space of feature terms as a directed graph (called the *subsumption graph*) where nodes are feature terms and directed edges indicate subsumption. Notice that the subsumption graph is a theoretical notion, completely defined by the subsumption relation, and that such graph is never explicitly constructed.

The two basic operations over the subsumption graph are unification and antiunification.

Definition 2. *(Unification) A* unification $\psi_1 \sqcup \psi_2$ *of two terms* ψ_1 *and* ψ_2 *is a most general term subsumed by both. A term* ψ *is called the* unifier *whenever:*

$$\begin{aligned}\psi_1 \sqcup \psi_2 = \psi : &(\psi_1 \sqsubseteq \psi \ \wedge\ \psi_2 \sqsubseteq \psi) \wedge \\ &(\nexists \psi' \sqsubset \psi :\ \psi_1 \sqsubseteq \psi' \ \wedge\ \psi_2 \sqsubseteq \psi')\end{aligned}$$

When two terms have contradictory information then they have no unifier — we write $\psi_1 \sqcup \psi_2 = \top$.

The *antiunification* ($\psi_1 \sqcap \psi_2$) of two terms ψ_1 and ψ_2 is defined as their *least general generalization* [18]:

Definition 3. *(Antiunification) An* antiunification $\psi_1 \sqcap \psi_2$ *of two terms* ψ_1 *and* ψ_2 *is a most specific term that subsumes both. The term is called the* antiunifier.

$$\begin{aligned}\psi_1 \sqcap \psi_2 = \psi : &(\psi \sqsubseteq \psi_1 \ \wedge\ \psi \sqsubseteq \psi_2) \wedge \\ &(\nexists \psi' \sqsupset \psi :\ \psi' \sqsubseteq \psi_1 \ \wedge\ \psi' \sqsubseteq \psi_2)\end{aligned}$$

Both unification and antiunification are operations over the subsumption graph: antiunification finds the most specific common "parent" (generalization); unification finds the most general common "descendant" (specialization). Moreover, unification and antiunification are not unique for the subsumption graph of feature terms. This means that the refinement graph of feature terms is not a lattice.

2.2. Refinement Operators

Let us now define the notion of *refinement operator* (for a more in depth discussion of refinement operators, see [12]), which can be used to navigate the subsumption graph, and, in general, any partially-ordered or quasi-ordered set. In the remainder of this article we will consider only the case of partially-ordered sets (i.e. in which two terms which subsume each other are considered equivalent: $\psi_1 \equiv \psi_2$).

Definition 4. *A* downward refinement operator ρ *over a partially-ordered set* $(\mathcal{L}, \sqsubseteq)$ *is a function such that* $\forall \psi \in \mathcal{L} : \rho(\psi) \subseteq \{\psi' \in \mathcal{L} | \psi \sqsubseteq \psi'\}$.

Definition 5. *An* upward refinement operator γ *over a partially-ordered set* $(\mathcal{L}, \sqsubseteq)$ *is a function such that* $\forall \psi \in \mathcal{L} : \gamma(\psi) \subseteq \{\psi' \in \mathcal{L} | \psi' \sqsubseteq \psi\}$.

Upward refinement operators generate terms which are more general, whereas downward refinement operators generate terms which are more specific. The symbol γ is used to symbolize upward refinement operators, and ρ to symbolize downward refinement operators. The following properties of refinement operators are considered desirable:

1. A refinement operator ρ is *locally finite* if $\forall \psi \in \mathcal{L} : \rho(\psi)$ is finite.
2. A downward refinement operator ρ is *complete* if $\forall \psi_1, \psi_2 \in \mathcal{L} | \psi_1 \sqsubseteq \psi_2 : \psi_2 \in \rho^*(\psi_1)$.
3. An upward refinement operator γ is *complete* if $\forall \psi_1, \psi_2 \in \mathcal{L} | \psi_1 \sqsubseteq \psi_2 : \psi_1 \in \rho^*(\psi_2)$.
4. ρ is *proper* if $\forall \psi_1, \psi_2 \in \mathcal{L}\ \psi_2 \in \rho(\psi_1) \Rightarrow \psi_1 \not\equiv \psi_2$.

where ρ^* stands for the *transitive closure* of a refinement operator. Intuitively, *locally finiteness* means that the refinement operator is computable, *completeness* means that all the terms in $\mathcal{L}$ can be generated by refinement, and *properness* means that a refinement operator does not generate elements which are equivalent to a given term ψ. A refinement operator is *ideal* when is locally finite, complete and proper. Other interesting properties, like *minimality* [4], have been discussed in the literature but are not relevant for this paper.

The *refinement graph* is the graph where each node is one term, and there is a link between two terms when one is a refinement of the other. Notice that the refinement graph is contained in the subsumption graph.

3. Properties and Disintegration

This section presents the *disintegration* operation, which disintegrates a given term (which might represent an instance or a generalization) into a set of properties. We introduced the idea of disintegration in our past work for the purposes of similarity assessment[2]. The intuitive idea of disintegration is that we want to transform a term into a set containing the most basic pieces of information the original term contains. For example, in the train shown in Figure 1, one property is that the train has 3 cars, another is that the first car has

[2]A less rigorous presentation of disintegration, without a discussion of it computational complexity, nor procedural definitions of the *remainder* operation can be found at [16].

4 wheels, another is that the number of wheels of the second car is the same as in the first car, etc. The disintegration operation provides with a formal and principled way to perform this process, based upon the existence of a refinement operator.

3.1. Properties

Disintegration splits a given term in a collection of smaller terms, called *properties*, representing its most primitive pieces of information. We will also show that, under certain assumptions, those properties can be integrated again to reconstruct the original example.

Let us first define the *remainder* of a generalization refinement operator.

Definition 6. *(Remainder) Given a term $\psi_2 \in \gamma(\psi_1)$, where γ is a generalization refinement, the* remainder *$r(\psi_1, \psi_2)$ of such generalization is a term π such that $\pi \sqcup \psi_2 \equiv \psi_1$ and $\nexists \psi' \in \mathcal{L}$ such that $\psi' \sqsubset \pi$ and $\psi' \sqcup \psi_2 \equiv \psi_1$.*

The remainder of a generalizing refinement γ from ψ_1 to ψ_2 is the most general term π such that when unified with the generalization ψ_2 obtains back the original term ψ_1. We will call this remainder π a *property* of ψ_1. Notice that the remainder is the most general term that captures which is the "property" that ψ_1 has and that is not present in ψ_2, i.e. the informational content that the generalization operator removed. Figure 2 illustrates this idea, where a train ψ_1 is generalized with a refinement operator to ψ_2: the property subtracted is the fact that the car of that train has 2 wheels. Notice that a property is, in general, a multi-relational pattern, with relations *cars* and *nwheels* in the property of Fig. 2. Computationally, a reminder term for a generalization refinement γ can be determined using Algorithm 1. This algorithm generalizes a seed term using γ to find a set of generalizations N such that satisfies the condition in step 3. One of this generalizations is selected at random an the process is iterated until it cannot longer find such a generalization, in which case the last generalization is the remainder. The remainder of a specialization refinement ρ can be defined similarly.

The computational complexity of Algorithm 1 for computing the remainder is $O(nm\Sigma + nm\Upsilon)$, where n is the length of the refinement path from ψ_1 to $\perp$, m is the average number of refinements generated by γ, Σ is the complexity of subsumption, and Υ is the complexity of unification, both of which are language dependent. In the case of feature terms, the cost of unification completely dominates that of subsumption [17], and thus, the cost of the remainder algorithm is basically $O(nm\Upsilon)$.

Algorithm 1 Remainder: $r(\psi_1, \psi_2, \gamma)$

```
1: $t := 0, \pi_0 := \psi_1$
2: while $(true)$ do
3:     $N = \{\psi \in \gamma(\pi_t) | \psi \not\sqsubseteq \psi_2 \wedge \psi \sqcup \psi_2 \equiv \psi_1\}$
4:     if $N = \emptyset$ then
5:         return $\pi_t$
6:     end if
7:     $\pi_{t+1}$ selected stochastically from $N$
8:     $t := t + 1$
9: end while
```

3.2. Disintegration

The intuitive idea of *term disintegration* is to generalize a term repeatedly until reaching $\perp$, collecting a property at each step by getting the remainder of the generalization operation. Let us formally define such process.

Definition 7. *(Refinement Path) A finite sequence of terms $(\psi_1, ..., \psi_n)$ is a* refinement path *$\psi_1 \xrightarrow{\rho} \psi_n$ between two terms ψ_1 and ψ_n when for each $1 \leq i < n$, $\psi_{i+1} \in \rho(\psi_i)$. The same definition applies for the generalization refinement operator: $\psi_n \xrightarrow{\gamma} \psi_1$.*

Definition 8. *(Disintegration) Given a finite refinement path $p = \psi_1 \xrightarrow{\gamma} \perp$ consisting of a sequence of terms $(\psi_1, ..., \psi_n)$, where $\psi_n = \perp$, the set $D_p(\psi_1) = \{r(\psi_i, \psi_{i+1})\}_{1 \leq i < n}$ is a* disintegration *of the term ψ_1.*

That is to say, $D_p(\psi_1)$ is the set of *remainders* resulting from each generalization step performed by the refinement operator γ in the path p from ψ_1 to $\perp$.

Given a refinement path $p = \psi \xrightarrow{\gamma} \perp$, and having in mind that refinement operators represent the most fine-grained steps in which terms can be specialized or generalized, the remainders obtained from such paths correspond to the most primitive pieces of information contained in a term ψ. Therefore, the disintegration of a term is a process that breaks up a term into its most constituent and primitive pieces of information (with respect to a particular language); each one of these pieces of information is also represented as a term, and this is what we call a *property*. Moreover, all the properties generated from a given refinement path are different as demonstrated by the following lemma.

Lemma 1. *If $\psi_1 \xrightarrow{\gamma} \perp$ is a refinement path consisting of a sequence of terms $(\psi_1, ..., \psi_n = \perp)$, γ is proper, and $1 \leq i < j < n$, then $r(\psi_i, \psi_{i+i}) \not\sqsubseteq r(\psi_j, \psi_{j+i})$.*

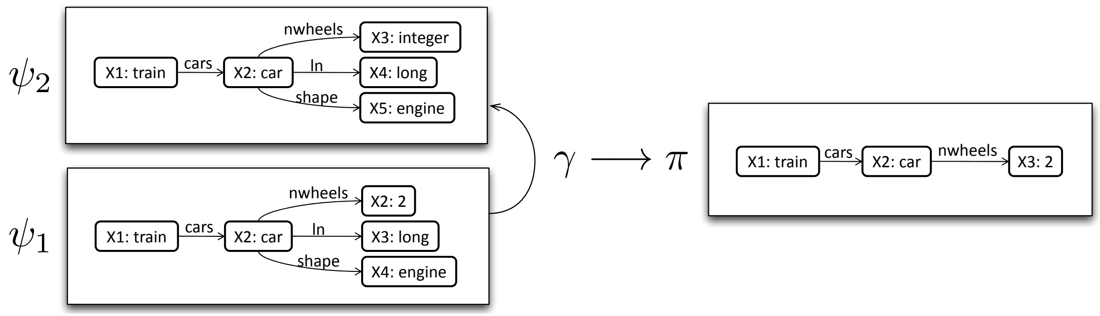


Fig. 2. A refinement operator γ that generalizes ψ_1 into ψ_2 by subtracting a piece of information π called the *remainder* of the refinement.

Algorithm 2 (Disintegrate): D(ψ, γ)

```
1: $D := \emptyset, t := 0, \psi_0 := \psi$
2: while $\psi_t \neq \perp$ do
3:     non-deterministically select a $\psi_{t+1} \in \gamma(\psi_t)$
4:     $D := D \cup \{r(\psi_t, \psi_{t+1})\}$
5:     $t := t + 1$
6: end while
7: return $D$
```

Proof. By Definition 7, we know that if γ is proper, then $\psi_{k+1} \sqsubset \psi_k$ for $1 \leq k < n$. Now, let $\psi_a = r(\psi_i, \psi_{i+i})$, and $\psi_b = r(\psi_j, \psi_{j+i})$. By Definition 6, $\psi_j \in \psi_{j+1} \sqcup \psi_b$, and thus $\psi_b \sqsubseteq \psi_j$, since $j > i$, we know that $\psi_j \sqsubseteq \psi_{i+1}$, and thus $\psi_b \sqsubseteq \psi_{i+1}$. Now, let us assume $\psi_a \sqsubseteq \psi_b$. Since $\psi_a \sqsubseteq \psi_b \sqsubseteq \psi_{i+1}$, we know that $\psi_{i+1} \sqcup \psi_b = \psi_{i+1}$, and $\psi_{i+1} \sqcup \psi_a = \psi_{i+1}$. Since γ is proper, we know that $\psi_i \neq \psi_{i+1}$. Therefore $\psi_i \notin \psi_{i+1} \sqcup \psi_a$, which is in contradiction with Definition 6, and thus our assumption that $\psi_a \sqsubseteq \psi_b$ has to be false. □

The disintegration of a term ψ is described in Algorithm 2. Given a term ψ, and a generalization refinement operator γ, the algorithm proceeds iteratively, generalizing ψ using γ, until ⊥ is reached. At each iteration t of the algorithm, a new generalization ψ_{t+1} is generated by taking one of the generalizations (one can be chosen at random) generated by γ from the current term ψ_t. Then, the property set D is expanded by adding the remainder $r(\psi_t, \psi_{t+1})$ of generalizing ψ_t into ψ_{t+1}. When ⊥ is reached, the algorithm returns the set D containing all the properties generated so far, corresponding to a disintegration of the term ψ.

Step 3 in Algorithm 2 is non-deterministic. Depending on the choice of refinements (i.e. depending on the refinement path), different disintegrations might be obtained. It can be shown that under certain conditions (e.g. the refinement graph being a lattice), then the choice of refinements is irrelevant, since all of them will result in the same disintegration. However, in general, this is not true. As part of our future work, we plan to investigate the possibility of different strategies for making this non-deterministic choice, which produce unique disintegrations.

Figure 3 shows an example of the disintegration process, where a simple train represented as a feature term (top half) has been disintegrated into properties (bottom half). Disintegration extracted 14 properties from this train. Notice how the properties respect Lemma 1, i.e. the properties generated earlier cannot be more general than the properties generated later.

Concerning computational complexity, the complexity of disintegration algorithm is $O(n^2 m \Upsilon)$, where n is the length of the refinement path from ψ to ⊥, m is the average number of refinements generated by γ, and Υ is the complexity of the unification operation.

The *integration* of a property set is the opposite process of disintegration.

Definition 9. *(Integration) The* integration *of a set of properties $D(\psi)$, obtained via disintegrating a term ψ is defined as:*

$$integrate(D(\psi)) = \bigsqcup(D(\psi))$$

In other words, integration is defined as the unification of all properties of a disintegrated term ψ. Given that unification of feature terms is not unique, there might be multiple different integrations of a given set of properties. It can be easily seen that one of the different integrations of a disintegrated term is equivalent to (in the sense of '≡') the original term.

Lemma 2. *One of the unifications of all the properties of a term ψ is exactly the term ψ, i.e. $\psi \in \bigsqcup(D(\psi))$. When unification is unique, then $\psi = \bigsqcup(D(\psi))$.*

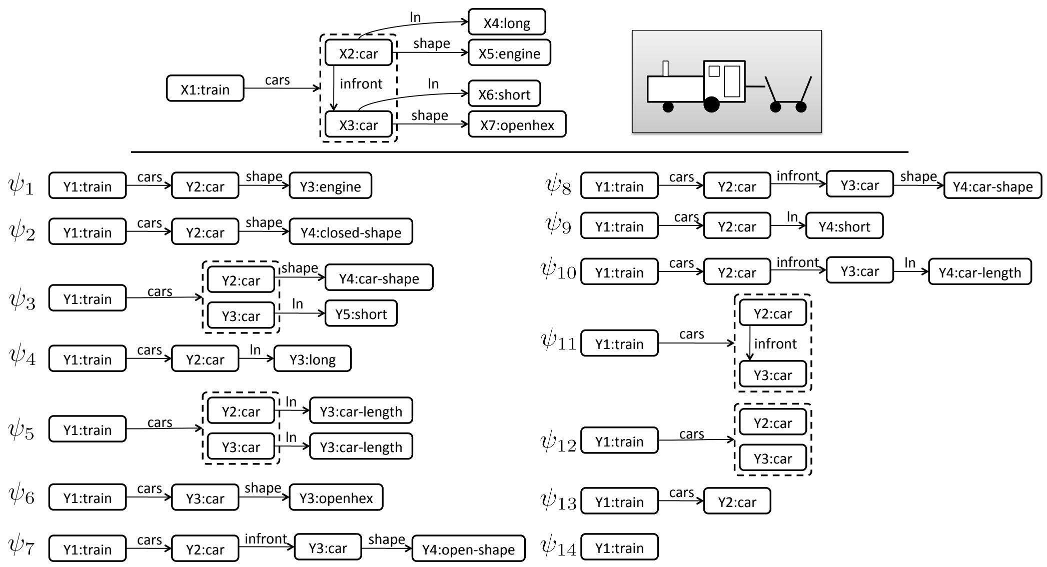

X1:train
cars
X2:car
X3:car
infront
ln
shape
X4:long
X5:engine
X6:short
X7:openhex
ψ1 Y1:train cars Y2:car shape Y3:engine
ψ2 Y1:train cars Y2:car shape Y4:closed-shape
ψ3 Y1:train cars Y2:car Y3:car shape Y4:car-shape ln Y5:short
ψ4 Y1:train cars Y2:car ln Y3:long
ψ5 Y1:train cars Y2:car Y3:car ln Y3:car-length ln Y3:car-length
ψ6 Y1:train cars Y3:car shape Y3:openhex
ψ7 Y1:train cars Y2:car infront Y3:car shape Y4:open-shape
ψ8 Y1:train cars Y2:car infront Y3:car shape Y4:car-shape
ψ9 Y1:train cars Y2:car ln Y4:short
ψ10 Y1:train cars Y2:car infront Y3:car ln Y4:car-length
ψ11 Y1:train cars Y2:car infront Y3:car
ψ12 Y1:train cars Y2:car Y3:car
ψ13 Y1:train cars Y2:car
ψ14 Y1:train

Fig. 3. An example feature term disintegrated into properties using Algorithm 2.

Proof. Let ψ_1 be a term that when disintegrated using the refinement path $(\psi_1, ..., \psi_n = \bot)$ yields the properties $(r_1, ... , r_{n-1})$. Definition 6 (Remainder) ensures that $\psi_{i+1} \sqcup r_i = \psi_i$ (or that ψ_i is one of the unifications if $\psi_{i+1} \sqcup r_i$ is not unique). Let us consider first the case when unification is unique. Iteratively unifying the properties in the reverse order in which they were generated, we can reconstruct the refinement path: $\psi_{n-1} = r_{n-1}$, $\psi_{n-2} = r_{n-1} \sqcup r_{n-2}$, $\psi_{n-3} = (r_{n-1} \sqcup r_{n-2}) \sqcup r_{n-3}$, etc. Thus, $\psi_1 = \bigsqcup_{i=n-1...1} r_i$, which is precisely $\bigsqcup(D(\psi))$. When unification is not unique, we know by Definition 6 that: $\psi_{n-1} = r_{n-1}$, $\psi_{n-2} \in r_{n-1} \sqcup r_{n-2}$, $\psi_{n-3} \in (r_{n-1} \sqcup r_{n-2}) \sqcup r_{n-3}$, etc. Thus, $\psi_1 \in \bigsqcup_{i=n-1...1} r_i$, which is precisely $\bigsqcup(D(\psi))$. □

If another representation formalism, different from feature terms, were to be used where the refinement graph was a lattice, then integration would generate a single term, which would be the same as the original example. This is interesting, since it shows that not too much information is lost when disintegrating a term.

4. Re-representation with a Taxonomic Vocabulary

The possibility of defining different refinement operators that generate different subsets of a given representation language opens the possibility of defining several different sublanguages of different expressive power and complexity. Consequently, the disintegration of the examples in a given training set would yield sets of properties in the particular sublanguage corresponding to the refinement operator being used.

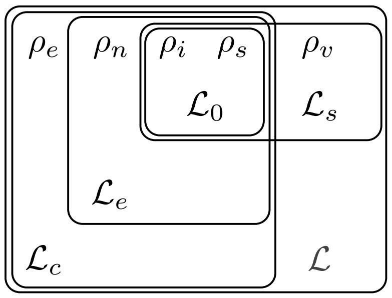
For example, Figure 4 shows the sublanguages defined in [16] for feature terms. Summarily, $\mathcal{L}$ is the complete feature term language as defined above; $\mathcal{L}_0$ contains all the feature terms that do not have any set-valued feature or any variable equality; $\mathcal{L}_e$ contains all the terms that do not have any set-valued feature or any circular variable equality (non-circular variable equalities are allowed); $\mathcal{L}_c$ is a super set of $\mathcal{L}_e$ which allows terms with circular variable equalities; and $\mathcal{L}_s$ is a super set of the base language $\mathcal{L}_0$ which allows set-valued features. These languages are generated using a different set of refinement operators (intuitive descriptions of each operator is included in the figure, see [16] for their formal definitions).

The disintegration operation gives us the capability of *re-representing examples* in a vocabulary composed of properties of those examples.

Definition 10. ***(Vocabulary)*** *A* vocabulary $\mathbf{V}$ *of properties for a set of examples* $E = \{e_1, \dots, e_n\}$ *is a subset* $\mathbf{V} \subseteq \bigcup_{i=1,\dots,n} D(e_i)$.

Definition 11. ***(Taxonomic Vocabulary)*** *A* taxonomic vocabulary *of a set properties* $\mathbf{V}$ *is the preorder* $\langle \mathbf{V}, \sqsubseteq \rangle$, *where* $\sqsubseteq$ *is the subsumption relation.*

Let us now show how a taxonomic vocabulary allows us to re-represent the set of examples $E = \{e_1, \dots, e_n\}$ as a binary matrix.



Refinement Operators:

ρ_e	Variable equality introduction
ρ_n	Non-circular variable equality introduction
ρ_i	Variable introduction (without sets)
ρ_s	Sort specialization
ρ_v	Variable introduction

Sub-languages:

$\mathcal{L}_0$	No sets, no variable equalities (Trees)
$\mathcal{L}_e$	No sets, no circular variable equalities (DAGs)
$\mathcal{L}_c$	No sets (Graphs)
$\mathcal{L}_s$	No variable equalities (Trees + sets)
$\mathcal{L}$	Full feature terms

Fig. 4. Several sublanguages of feature terms generated by different refinement operators (shown as ρ symbols on the top and defined in [16]).

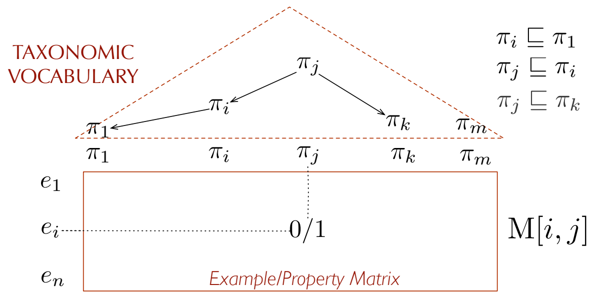

TAXONOMIC VOCABULARY
π_j
π_i
π_k
π_1
π_m
$\pi_i \sqsubseteq \pi_1$
$\pi_j \sqsubseteq \pi_i$
$\pi_j \sqsubseteq \pi_k$
π_1 π_i π_j π_k π_m
e_1
e_i
e_n
0/1
M$[i,j]$
Example/Property Matrix

Fig. 5. Taxonomic vocabulary of properties to re-represent examples and the E/M matrix.

Definition 12. ***(Re-representation)*** *A re-representation of a set of examples $E = \{e_1, \ldots, e_n\}$ with a taxonomic vocabulary $\langle \mathbf{V}, \sqsubseteq \rangle$, where $\mathbf{V} = \{\pi_1, ..., \pi_m\}$, is a $n \times m$ binary matrix* **M** *where*

$$\mathbf{M}[i,j] = \begin{cases} 1 \Leftrightarrow \pi_j \sqsubseteq e_i \\ 0 \Leftrightarrow \pi_j \not\sqsubseteq e_i \end{cases}$$

We call **M** the Example/Property (or E/P) matrix, as shown in Fig. 5.

Definition 13. ***(Example Re-representation)*** *The re-represenation of an example e_i in a taxonomic vocabulary $\langle \mathbf{V}, \sqsubseteq \rangle$ is a Boolean vector $R(e_i) = (b_1, \cdots, b_m)$, where $b_i \in \{0,1\}$ such that $b_j = 1$ whenever $\pi_j \sqsubseteq e_i$ and $b_j = 0$ otherwise. Notice that each row of the E/P matrix is an example re-representation.*

Thus, given a dataset E, we can re-represent it by first disintegrating each of the examples in E, and then computing the E/P matrix. Moreover, a vocabulary **V** may also be built using only a subset of all the available examples in E. A simple way to do that is sampling the examples to be disintegrated. Let $\Sigma(E, \tau)$ be a sampling method (such as SRS or class-stratified sampling) that returns a τ percent of E; the corresponding vocabulary is $\mathbf{V}_{\Sigma(E,\tau)} = \bigcup_{e_i \in \Sigma(E,\tau)} D(e_i)$. For large data sets, sampling the examples to collect the properties for a vocabulary clearly diminishes computational cost, as long as the τ percent examples allow us to build a vocabulary **V** that is representative and satisfactory for the purposes at hand.

There are further operations that may reduce the size of a vocabulary **V**, such are removing all properties subsumed by all (or almost all) examples. But are not studied in this paper.

Re-representation is a process that maps objects described in a formalism to descriptions on another formalism, often because this second formalism is more adequate for some specific form of reasoning or inference (e.g. analogical reasoning [8]). A related notion is that of propositionalization in relational learning and ILP [11]; from our viewpoint, propositionalization is a specific kind of re-representation where relational representation of objects is mapped into a propositional language. In our approach, however, objects represented as feature terms are mapped not onto propositions but onto sets of feature terms (called properties). These feature terms (properties) constitute a partially ordered vocabulary whose elements are not simple propositions, since they have a strong structure based on subsumption. Moreover, properties are also related to the objects (examples) by subsumption.

Let us now analyze the structure of taxonomic vocabularies that are generated for different data sets.

5. Structure of Taxonomic Vocabularies

The disintegration operation maps a given relational example (represented, for example, as a feature term) to a set of properties. Moreover, these properties are not completely independent, but are related to each other via the subsumption relation. For example, in Figure 3, property ψ_{14} subsumes property ψ_{13}. In a similar way, a taxonomic vocabulary **V** is not a plain

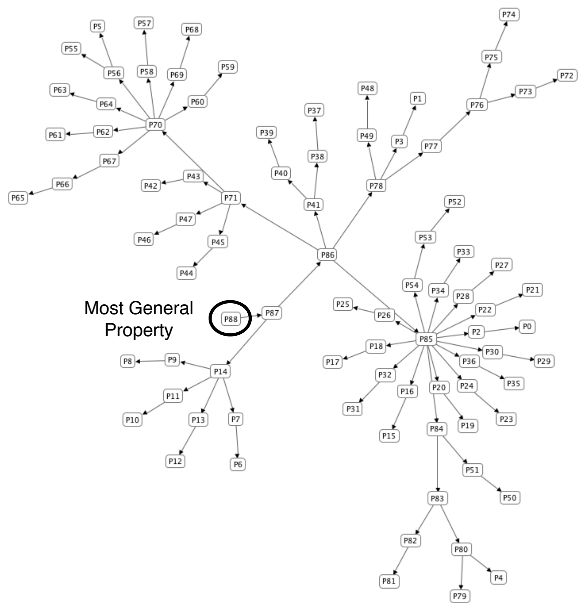


Fig. 6. The set of properties resulting from disintegrating an example in the Demospongiae data set. Arrows indicate subsumption.

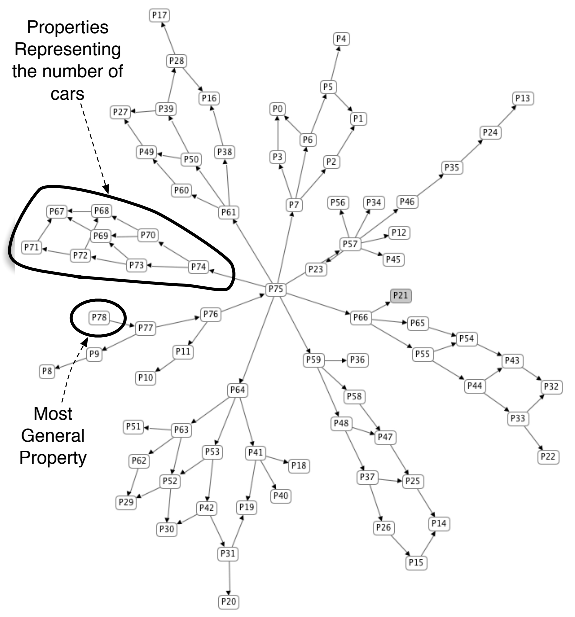


Fig. 7. The set of properties resulting from disintegrating an example in the Trains data set. Arrows indicate subsumption.

set, but has internal structure (see Definition 11). This section illustrates the structure of these vocabularies for different types of datasets.

Figure 6 shows the set of 88 properties resulting from disintegrating one of the examples in the Demospongiae dataset. Demospongiae is a relational dataset containing descriptions of marine sponges. These descriptions are tree-like structures (i.e. a feature term representing a sponge might have set-valued features, but no variable equalities). If we disintegrate one of these sponges and create a graph where each node is one of the resulting properties and edges represent subsumption relations, we obtain a graph similar to the one shown in Figure 6. We call this the *property graph* of a given feature term. Notice that since subsumption is transitive, if a property ψ_1 subsumes a property ψ_2, which subsumes a property ψ_3, we only draw edges between ψ_1 and ψ_2 and between ψ_2 and ψ_3; we do not include the edge between ψ_1 and ψ_3, which is implicit. Taxonomic vocabularies for feature terms represented by feature term sub-languages without variable equalities result in trees, where the root is the most general property.

Figure 7 shows the property graph resulting from disintegrating an example in the Trains data set (where each train is represented similarly as the train in Figure 1). To represent this dataset, we need variable equalities, but not circular ones. The resulting structure of the property graph is a DAG (directed acyclic graph). Moreover, we can see that the resulting DAG is organized by smaller clusters of properties; in Figure 7, we have highlighted one of these clusters, containing properties that represent the number of cars in the train. For example:

- P75 corresponds to ψ_{14} in Figure 3 (representing a train with no cars).
- P74 corresponds to ψ_{13} (a train with one car).
- P73 is ψ_{12} (a train with two cars, but without any order restriction between them).
- P69 is ψ_{11} (a train with two cars, one in front of the other).

Notice that, by construction, the different properties in the re-representation of examples are not independent, as typically assumed by machine learning algorithms. Thus, it is interesting to know the structure of the property graph associated with the Taxonomic vocabularies for a given representation language, since this structure is tightly related with the dependencies between attributes. If we re-represent examples as binary vectors, the binary features that are related via subsumption are not independent. This information can be exploited for example for feature selection, or for increasing the efficiency of algorithms like decision tree inducers (ID3, C4.5, FOIL, etc.), since once a feature is selected, all of the features subsumed by it can be discarded in one of the subbranches of the tree. As part of our future work, we want to explore new algorithms that can further exploit this structure during learning.

6. Experimental Results

In order to evaluate our re-representation ideas, we performed two sets of experiments. In the first experiment, we compared the performance of a nearest-neighbor classifier using distance measures based on our re-representation versus a set of other relational distance measures that work directly over the relational representation of the examples. In the second experiment, we studied the effect that creating taxonomic vocabularies of different sizes has on performance.

We used three datasets, representing three levels of complexity: Soybean is a propositional data set (representable using $\mathcal{L}_0$ in Figure 4) from the UCI machine learning repository consisting of 307 cases. Demospongiae is a relational dataset, where each example can be represented as a tree ($\mathcal{L}_s$); it has 503 examples pertaining to 7 different classes (we used also a subset of 280 examples and 3 different classes, typically used in the literature for comparison purposes). Trains is a relational dataset, generated using Muggleton's train generator [14]. It contains 1000 trains belonging to 2 different classes; trains are represented as graphs (the full feature term language $\mathcal{L}$ is required for this dataset).

We experimented with two different distance measures for the nearest-neighbor classifier: Jaccard and Euclidean. These two measures are defined as measures that compare two rows, i and j (representing two instances), of the re-representation matrix **M**, as follows (assuming we have n examples and m properties):

- **Jaccard distance**: typically defined to measure the difference between sets as the size of their intersection divided by the size of their union, can be computed from the matrix **M** as:

$$d_J(i,j) = \frac{\sum_{k=1\ldots m} M[i,k]M[j,k]}{\sum_{k=1\ldots m} M[i,k] + M[j,k] - M[i,k]M[j,k]}$$

- **Euclidean distance**:

$$d_E(i,j) = \sqrt{\sum_{k=1\ldots m} (M[i,k] - M[j,k])^2}$$

The results from our first experiment can be seen on Table 1. We compared the performance of our re-representation against four other distance and similarity measures from the literature:

- S_λ [16]: is a relational similarity measure based on computing the antiunification of the two terms, and then measuring its size (the larger, the more similar the two terms).
- SHAUD [3]: is a similarity measure defined over feature terms. SHAUD is only defined for terms that can be represented as trees, and thus, it cannot be applied to the trains dataset.
- RIBL [7]: is a relational similarity measure defined over horn clauses. In order to compare against this measure, we converted all our datasets to Horn clauses, using the same procedure as [16].
- Kashima [9]: is a graph-kernel based on random walks. It requires the conversion of the examples to labelled graphs, which can be done trivially.

As Table 1 shows, using our re-representation achieves similar (and sometimes better) performance than using complex relational similarity measures. This indicates that our re-representation does not lose any significant information, at least in the datasets used for this study. Moreover, as any propositionalization technique, it has the advantage that once a dataset has been re-represented, applying propositional machine learning techniques is much more efficient.

Table 2 show the results we obtained disintegrating a different portion of examples (τ) of the dataset. Results were obtained using a leave-one-out procedure. As Table 2 shows, performance degrades very slowly when disintegrating smaller and smaller portions of a regular dataset. For example, we observed no difference in performance in any dataset when creating the Taxonomic Vocabulary by disintegrating only 40% of the examples in the dataset. Moreover, in some datasets, such as Trains, performance did not degrade at all even when only disintegrating a 1% of the examples in the dataset. This is because just disintegrating a small sample of examples in the dataset is enough for obtaining a good sample of binary attributes that can be used to perform the classification task.

These results are very promising, since disintegrating is a computationally expensive operation, but as Table 2 shows, we only need to disintegrate a small portion of the dataset for obtaining good results. Also, notice that when classifying new instances, we don't need to disintegrate them, just re-represent them, which is computationally much cheaper than disintegration. Re-representing simply means running one subsumption test for each of the different properties in the Taxonomic Vocabulary.

Table 1
Classification accuracy (in percentage) measured using a leave-one-out method for different distance/similarity measures.

	Using re-representation		Relational Measures			
	Jaccard	Euclidean	S_λ	SHAUD	RIBL	Kashima
Soybean-307	91.21	91.21	91.53	91.53	91.53	92.18
Demospongiae-280	95.00	93.47	95.00	95.71	91.67	90.71
Demospongiae-503	90.66	89.26	89.66	88.27	88.93	83.10
Trains-1000	90.70	90.40	84.80	-	67.10	63.90

Table 2
Performance of a nearest neighbor classifier with different sampling sizes (τ) for vocabulary creation; for each value of τ we report classification accuracy of three different distances and vocabulary size.

	Soy (307)			Dem (280)		
τ	Jacc.	Eucl.	Size	Jacc.	Eucl.	Size
0.01	86.97	86.32	92.04	88.57	87.50	95.39
0.02	85.99	86.64	107.08	88.21	86.79	140.55
0.05	90.55	90.88	122.00	93.21	90.71	197.80
0.1	90.55	90.55	129.52	93.57	92.50	247.81
0.2	90.88	90.88	133.11	93.93	93.93	306.56
0.4	91.53	91.53	134.53	95.00	93.93	376.55
0.6	90.88	90.88	134.82	95.36	93.57	428.26
0.8	90.88	90.88	134.94	95.00	94.07	469.86
1.0	91.21	91.21	135.00	95.00	93.47	506.39
	Dem (503)			Trains (1000)		
τ	Jacc.	Eucl.	Size	Jacc.	Eucl.	Size
0.01	83.50	83.10	149.81	92.00	91.40	86.68
0.02	86.68	86.68	197.55	90.90	90.40	99.08
0.05	89.46	88.87	271.67	91.40	91.20	116.42
0.1	89.46	89.06	335.93	91.40	91.70	129.44
0.2	90.86	89.46	408.60	91.40	91.20	141.11
0.4	90.66	90.46	497.42	90.30	89.80	150.22
0.6	90.66	90.06	560.37	91.20	90.30	154.40
0.8	90.46	90.06	609.30	90.80	90.90	156.70
1.0	90.66	89.26	651.61	90.70	90.40	158.00

7. Related Work

Propositionalization in ILP transforms a relational representation of a learning problem into a propositional (feature-based, attribute-value) representation [11]. Clearly, propositionalization can be described as a re-representation that transforms examples into vectors of attribute-value pairs, while in our approach the re-representation transforms examples into sets of properties (multi-relational patterns, not attribute-value pairs). An example of propositional learning techniques applied to relational data is adapting decision trees to ILP, e.g. the TILDE system [5]. TILDE presents a logical representation for decision trees and how to translate them into Prolog programs, and uses first order logic clauses to represent decisions (nodes) in the tree. MRDT (Multi-Relational Decision Tree Induction) were introduced in [10]. MRDT adds decision nodes to the tree through a process of successive refinement. MRDT defines a so-called *selection graph* where each node contains a multi-relational pattern, and a new split in the tree modifies ("refines") the selection graph. While properties are also multi-relational patterns, they are defined by and obtained from the *refinement graph* of the generalization space, not a selection graph. Neither TILDE nor MRDT use re-representation of examples, and they focus on logical representations, while our approach can be used in any representation formalism where an adequate refinement operator could be defined.

8. Conclusions

This paper is based upon two key ideas. First we presented the idea of *disintegration* (already introduced in [16]), which allows us to translate a term into a set of properties. The second is that of a taxonomic vocabulary, which allows us to define an E/P matrix **M** relating properties to examples can be used for any classical approach of propositional learning to relational data. The reason is that the E/P matrix **M** re-represents the relational examples into a Boolean vector of properties that each example satisfy or not. The only difference is that the values in that vector are relational patterns (properties) that can be obtained in a principled way after a refinement graph is specified. The E/P matrix **M** could be used for different purposes, like calculating the similarity among examples, using **M** to perform clustering, or finding the minimal set of properties that discriminate the positive examples belonging to a class with respect to the negative examples.

We presented experiments of using re-representation in the context of a nearest neighbor classifier. In our current work, we are developing property-based decision trees. One interesting aspect of our re-representation approach when used in conjunction with inductive machine learning techniques, such as decision trees, is that once generalizations have been found over the re-representation, those can be mapped back to the original representation by using the *integration* operation.

Future work will also explore how to use the various distinct sublanguages (e.g. like the sublanguages of feature terms defined in Fig. 4) in machine learning techniques. The goal would be to find the simplest sublanguage that fulfills the requirement of a specific ML technique for a given set of examples (following ideas from multi-strategy learning). Moreover, we intend to investigate the use of properties on other representation formalisms for which complete and locally finite refinement operators have been defined in the literature, such as ILP-restricted Horn Clauses [12] or some Description Logics [4].

Acknowledgements. Research partially supported by Projects Next-CBR (TIN2009-13692-C03-01) and Cognitio (TIN2012-38450- C03-03) and by the Generalitat de Catalunya under the grants 2009-SGR-1434.

References

[1] Aït-Kaci, H.: Description logic vs. order-sorted feature logic. In: Proc. 20th International Workshop on Description Logics. pp. 147–154 (2007)

[2] Aït-Kaci, H., Podelski, A.: Towards a meaning of LIFE. Tech. Rep. 11, Digital Research Laboratory (1992)

[3] Armengol, E., Plaza, E.: Relational case-based reasoning for carcinogenic activity prediction. Artif. Intell. Rev. 20(1-2), 121–141 (2003)

[4] Badea, L., Nienhuys-Cheng, S.H.: A refinement operator for description logics. In: Cussens, J., Frisch, A. (eds.) Inductive Logic Programming. pp. 40–59. No. 1866 in Lecture Notes in Computer Science, Springer (1999)

[5] Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision trees. Artificial Intelligence 101, 285–297 (May 1998)

[6] Carpenter, B.: The Logic of Typed Feature Structures, Cambridge Tracts in Theoretical Computer Science, vol. 32. Cambridge University Press (1992)

[7] Emde, W., Wettschereck, D.: Relational instance based learning. In: Saitta, L. (ed.) Machine Learning - Proceedings 13th International Conference on Machine Learning. pp. 122 – 130. Morgan Kaufmann Publishers (1996)

[8] Falkenhainer, B., Forbus, K., Gentner, D.: The structure mapping engine: algorithm and examples. Artificial Intelligence 41, 1–63 (1990)

[9] Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: Proc. ICML'04. pp. 321–328 (2003)

[10] Knobbe, A.J., Siebes, A., Wallen, D.V.D., V, S.B.: Multi-relational decision tree induction. In: Principles of Data Mining and Knowledge Discovery. pp. 378–383. Springer (1999)

[11] Kramer, S., Lavrač, N., Flach, P.: Propositionalization approaches to relational data mining. In: Džeroski, S., Lavrač, N. (eds.) Relational Data Mining, pp. 262–286. Springer (2000)

[12] van der Laag, P.R.J., Nienhuys-Cheng, S.H.: Completeness and properness of refinement operators in inductive logic programming. J. Log. Program. 34(3), 201–225 (1998)

[13] Lavrač, N., Džeroski, S.: Inductive Logic Programming. Techniques and Applications. Ellis Horwood (1994)

[14] Michie, D., Muggleton, S., Page, D., Srinivasan, A.: To the international computing community: A new East-West challenge. Tech. rep., Oxford University Computing laboratory, Oxford,UK (1994)

[15] Mitchell, T.: Machine Learning. McGraw-Hill (1997)

[16] Ontañón, S., Plaza, E.: Similarity measures over refinement graphs. Machine Learning 87(1), 57–92 (Apr 2012)

[17] Ontañón, S., Meseguer, P.: Efficient operations in feature terms using constraint programming. In: Inductive Logic Programming, pp. 270–285. Springer (2012)

[18] Plotkin, G.D.: A note on inductive generalization. In: Machine Intelligence. No. 5 (1970)

[19] Sánchez, A., Ontañón, S., Calero, P.A.G., Plaza, E.: Measuring similarity in description logics using refinement operators. In: Ram, A., Wiratunga, N. (eds.) ICCBR'11: Proc. 19th International Conference on Case-Based Reasoning. Lecture Notes in Artificial Intelligence, vol. 6880, pp. 289 – 303 (2011)

[20] Sánchez-Ruiz-Granados, A.A., Ontañón, S., González-Calero, P.A., Plaza, E.: Refinement-based similarity measure over DL conjunctive queries. In: Delany, S.J., Ontañón, S. (eds.) ICCBR-13: Proc. 21st International Conference on Case-Based Reasoning. Lecture Notes in Computer Science, vol. 7969, pp. 270–284. Springer (2013)