An Approach to Re-representation in Relational Learning

Santiago Ontañón [a] and Enric Plaza [b]
[a] *Computer Science Department, 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

Abstract. 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.

1. Introduction

Relational Machine Learning (RLM) studies how to design machine learning algorithms for domains where data is structured, or relational. In order to address this problem, in this paper 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 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[1].

Our approach is based on the notion of *refinement graphs* [9], which are used in ILP and other techniques performing inductive relational learning, e.g in Description Logics [3,11,12]. A refinement graph in the space of generalizations is built by defining a suitable *refinement operator*. The approach and techniques presented here are, in principle, applicable to any refinement graph where the refinement operator satisfies certain properties, namely they have to be complete and locally finite. For reasons of space an clarity, however, this paper will focus on

[1] This paper has been selected by the CCIA'2013 scientific committee for an extended version in a special issue on AICommunications journal

a particular representation formalism, namely *feature terms* (presented in the next section), for which we will define its own refinement graph. Then, we present the notion of a *property* (a multi-relational pattern) of a relational example, and the *disintegration* 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 6.

2. Preliminaries

This section introduces the formalism of feature terms, and the basic notions of refinement operators and graphs.

2.1. Feature Terms

Feature terms [2,5] (also called 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 $\bot$ 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 $\mathcal{S}$, where $s \leq s'$ 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 either a variable $Y \in \mathcal{V}$, or a set of variables $\{X_1, ..., X_m\}$.

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): *cars* of X_1. 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[2].

[2]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.

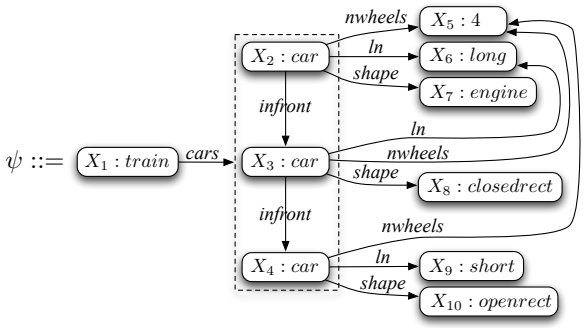

Figure 1. A train represented as a feature term.

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 formed given a signature, and that contain the infimum $\bot$ 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.

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

Definition 2 *(Unification) The* unification $\psi_1 \sqcup \psi_2$ *of two terms* ψ_1 *and* ψ_2 *is the 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 — equivalently 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*:

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

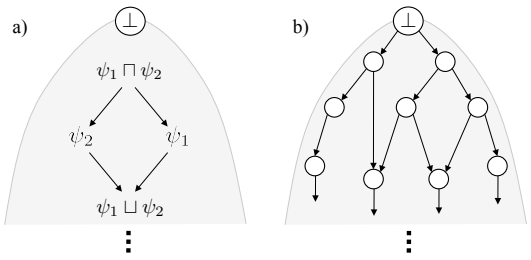

Figure 2. a) Illustration of the unification and antiunification concepts, b) refinement graph.

$$\psi_1 \sqcap \psi_2 = \psi : (\psi \sqsubseteq \psi_1 \ \wedge \ \psi \sqsubseteq \psi_2) \wedge \\ (\nexists \psi' \sqsupset \psi : \ \psi' \sqsubseteq \psi_1 \ \wedge \ \psi' \sqsubseteq \psi_2)$$

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. Figure 2.a illustrates unification and antiunification, showing the entire space of feature terms (with the most general term, $\perp$, at the top) and arrows indicating subsumption.

2.2. Refinement Operators

Let us now define the notion of *refinement operator* (for a more in depth discussion of refinement operators, see [9]), 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* $\forall \psi \in \mathcal{L} : \gamma(\psi) \subseteq \{\psi' \in \mathcal{L} | \psi' \sqsubseteq \psi\}$.

In other words, upward refinement operators generate terms which are more general, whereas downward refinement operators generate terms which are more specific. Typically, the symbol γ is used to symbolize upward refinement operators, and ρ to symbolize either a downward refinement operator, or a refinement operator in general. 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.

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. Figure 2.b illustrates a refinement graph, where more general terms are drawn at the top, and arrows indicate specialization refinement. 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 (a generalization or an example) into a set of properties. We introduced the idea of disintegration in our past work for the purposes of similarity assessment (a more informal definition can be found in [10]). The intuitive idea of disintegration is that we want to disintegrate a term into the most basic pieces of information it 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 4 wheels, another is that the number of wheels of the second car is the same as in the first car, etc.

3.1. Properties

The disintegration operation defined in this section specifically splits a given term in a collection of smaller terms, called *properties*, representing precisely these most primitive pieces of information. We will also show that, under certain assumptions, those properties can be integrated again to reconstruct the original example. Those pieces of information will then be used as the individual features to be considered when learning subsumption trees.

Before defining the properties of a term, we will 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 $\psi' \sqcup \psi_2 \equiv \psi_1$.*

That is to say, 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 3 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. 3. The remainder of a specialization refinement ρ can be defined similarly.

Algorithm 1 (Disintegrate): D(ψ, γ)

1: $D := \emptyset, t := 0, \psi_0 := \psi$
2: **while** $\psi_t \neq \bot$ **do**
3: $\quad \psi_{t+1} \in \gamma(\psi_t)$ (random selection)
4: $\quad D := D \cup \{r(\psi_t, \psi_{t+1})\}$
5: $\quad t := t + 1$
6: **end while**
7: **return** D

3.2. Disintegration

Now, if we iterate this generalization refinement over the resulting term and keep generalizing it, we will obtain a collection of properties as remainders of each step. In the end, the iterative generalization process will reach $\bot$, the empty term, and we will have a collection of properties satisfied by the initial term. This is the intuitive idea of *term disintegration*: generalize a term repeatedly until reaching $\bot$ while collecting a property at each step by getting the remainder of the generalization operation.

Definition 7 *(Disintegration) Given a finite refinement path* $p = \psi_1 \xrightarrow{\gamma} \bot$ *consisting of a sequence of terms* $(\psi_1, ..., \psi_n = \bot)$*, the set* $D_p(\psi_1) = \{r(\psi_i, \psi_{i+i})\}_{1 \leq i < n}$ *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 $\bot$.

Given a refinement path $p = \psi \xrightarrow{\gamma} \bot$, 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*.

The disintegration of a term ψ is described in Algorithm 1. Given a term ψ, and a generalization refinement operator γ, the algorithm proceeds iteratively, generalizing ψ using γ, until $\bot$ 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 $\bot$ is reached, the algorithm returns the set D containing all the properties generated so far, corresponding to a disintegration of the term ψ.

Notice that step 3 in Algorithm 1 is non-deterministic, since any refinement can be chosen. This means, that depending on the choice of refinements, 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 disintegration definitions, which produce unique disintegrations.

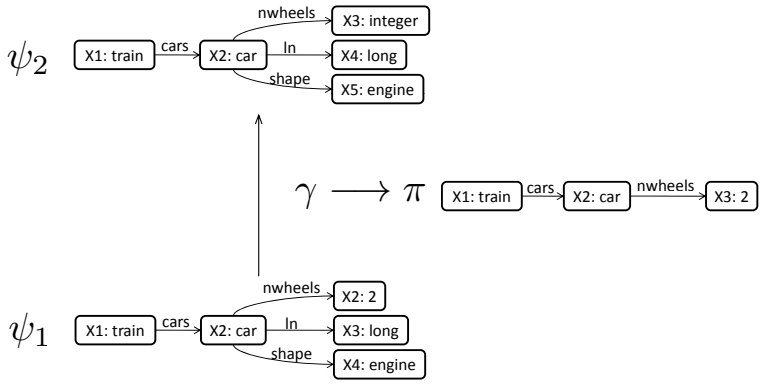

Figure 3. A refinement operator γ that generalizes ψ_1 into ψ_2 by subtracting a piece of information ψ called the *remainder* of the refinement.

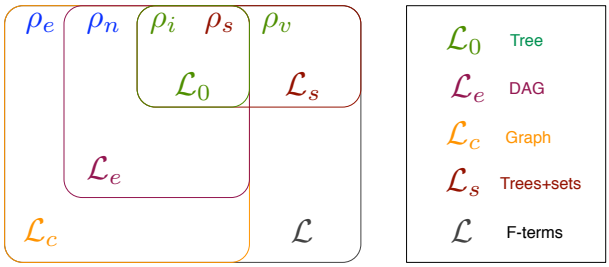

Figure 4. Several sublanguages of feature terms generated by different refinement operators (shown as ρ symbols on the top and defined in [10]).

The *integration* of a property set is the opposite process of disintegration. Defined as $integrate(D(\psi)) = \bigsqcup(D(\psi))$, it means 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 '$\equiv$') the original term. If another representation formalism, different from feature terms, were to be used where the refinement graph was to a partial order but a lattice, then integration would generate a single term, which would be the same (in the sense of '$\equiv$') as the original example.

4. Re-representation with a Taxonomic Vocabulary

The possibility of defining different refinement operators opens the possibility of defining several different sublanguages of different expressive power and complexity. Consequently, disintegration of examples would yield sets of properties in the particular sublanguage corresponding to the refinement operator being used. Figure 4 shows the sublanguages defined in [10] for feature terms. Summarily, $\mathcal{L}$ is the complete feature term as defined here; $\mathcal{L}_0$ 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.

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

Definition 8 (Vocabulary) *A* vocabulary $\mathbf{V}$ *of properties for a set of examples* $E = \{e_1, \ldots, e_n\}$ *is a subset* $\mathbf{V} \subseteq \bigcup_{i=1,\ldots,n} D(e_i)$.

Definition 9 (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 between properties.*

Notice that the union of properties means that there will be no "repeated" properties in $\mathbf{V}$ —i.e. if two properties are equivalent only one is in $\mathbf{V}$. This allows us to re-represent the set of examples $E = \{e_1, \ldots, e_n\}$ as a binary matrix.

Definition 10 (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* $\mathbf{M}$ *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 $\mathbf{M}$ the Example/Property (or E/P) matrix, as shown in Fig. 5.

Definition 11 (Example Re-representation) *The re-represenation of an example* e_i *in a taxonomic vocabulary* $\langle \mathbf{V}, \sqsubseteq \rangle$ *is a Boolean vector* $R(e_i) = (b_1, \cdots, b_m) \in \{0,1\}^m$ *such that* $b_j = 1$ *whenever* $\pi_j \sqsubseteq e_i$ *and* $b_j = 0$ *otherwise.*

Moreover, a vocabulary $\mathbf{V}$ may also be built using only a subset of all the available examples $E = \{e_1, \ldots, e_n\}$. 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 $\mathbf{V} = \bigcup_{e_i \in \Sigma(E,\tau)} D(e_i)$. 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 $\mathbf{V}$ that is representative and satisfactory for the purposes at hand.

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 [6]). A related notion is that of propositionalization in relational learning and ILP [8]; 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.

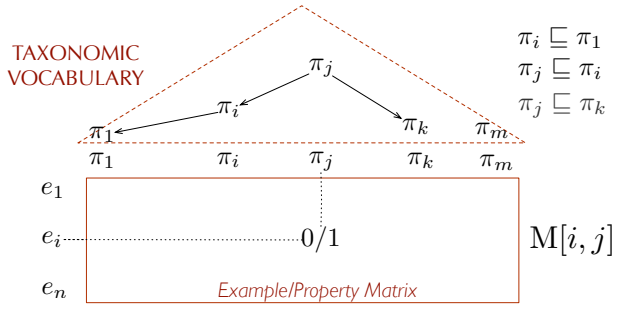

Figure 5. Taxonomic vocabulary of properties to re-represent examples and the E/M matrix.

Moreover, properties are also related to the objects (examples) by subsumption, which is used in the following section to learn binary trees.

5. Related Work

Propositionalization in ILP transforms a relational representation of a learning problem into a propositional (feature-based, attribute-value) representation [8]. 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 [4]. 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 [7]. 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 graph could be defined.

6. Conclusions

This paper has introduced two key ideas. First we presented the idea of *disintegration*, which allows us to define a property-based vocabulary and to re-represent relational examples in this vocabulary. The second is that the 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 $\mathbf{M}$ could be used for different purposes, like calculating the similarity among examples, using $\mathbf{M}$ 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.

These two approaches are as yet future work, but re-representation into a property-based vocabulary has been used by the authors in estimating similarity for k-Nearest Neighbour [10] and we are developing property-based decision trees. Future work will also explore how to use the several distinct sublanguages $\mathcal{L}_x$ of 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. 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 [9] or some Description Logics [3].

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] 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)
[4] Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision trees. Artificial Intelligence 101, 285–297 (May 1998)
[5] Carpenter, B.: The Logic of Typed Feature Structures, Cambridge Tracts in Theoretical Computer Science, vol. 32. Cambridge University Press (1992)
[6] Falkenhainer, B., Forbus, K., Gentner, D.: The structure mapping engine: algorithm and examples. Artificial Intelligence 41, 1–63 (1990)
[7] 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)
[8] 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)
[9] 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)
[10] Ontañón, S., Plaza, E.: Similarity measures over refinement graphs. Machine Learning 87(1), 57–92 (Apr 2012)
[11] 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)
[12] 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)