

Formulating Description Logic Learning as an Inductive Logic Programming Task

Stasinos Konstantopoulos and Angelos Charalambidis

Abstract—We describe an Inductive Logic Programming (ILP) approach to learning descriptions in Description Logics (DL) under uncertainty. The approach is based on implementing many-valued DL proofs as propositionalizations of the elementary DL constructs and then providing this implementation as background predicates for ILP. The proposed methodology is tested on a many-valued variation of *eastbound-trains* and *Iris*, two well known and studied Machine Learning datasets.

I. INTRODUCTION

Description logics (DL) are a family of logics that has found many applications in conceptual and semantic modelling, and is one of the key technologies behind *semantic web* applications.

Fuzzy and, in general, many-valued extensions of DL semantics have given significant boost to their importance in both related fields: semantic conceptualization gains a means of expressing the uncertainty that is inherent in real-world modelling problems and uncertainty inference gains access to the vast conceptualization effort that has been carried out in the context of the semantic web.

Despite, however, the rapid progress in inference methods for many-valued DL, there has been very limited success in applying machine learning methodologies to this family of logics, and especially to its more expressive members (such as those covering OWL and OWL 2) that are routinely used in web intelligence applications.

In this paper we first introduce the machine learning discipline of Inductive Logic Programming and then discuss previous work on applying ILP to learning DL (Section II). These approaches tend to propose adaptation of ILP algorithms so that they cover DL, but are restricted to the less expressive members of the DL family. Instead, we investigate a novel approach whereby we re-formulate DL inference within the ILP paradigm, effectively mapping our problem to an equivalent problem within the domain of application of ILP (Section III). We evaluate our approach by applying an unadapted ILP system to a DL learning task under this mapping (Section IV), and close the paper by drawing conclusions and outlining future research directions (Section V).

II. BACKGROUND

In this section we shall briefly discuss Inductive Logic Programming and previous work on its application to the synthesis of Description Logic knowledge, both crisp and many-valued.

Both authors are with the Institute of Informatics and Telecommunications, NCSR 'Demokritos', Aghia Paraskevi 153 10, Athens, Greece (email: {konstant,acharal}@iit.demokritos.gr).

A. Inductive Logic Programming

Inductive logic programming (ILP) is the machine learning discipline that lies at the intersection between inductive machine learning and logic programming: the task of ILP is the construction of *hypotheses* such that, given prior *background knowledge*, they explain or predict a set of empirical observations.

ILP literature is rich in alternative specifications of this general task definition, each with its own strengths and weaknesses in terms of applicability to real problems and efficiency of the algorithms that approach it. We direct the interested reader elsewhere for fuller discussions of ILP and its various ILP settings [1] and focus on the most commonly used *example setting* under *definite semantics*: under definite semantics ILP both background and constructed hypotheses are definite Horn clauses and in the example setting of this semantics the observations are further restricted to fully ground positive and negative examples.

The importance of this setting is that key theoretical results and algorithms have been established within it, including the reduction of the task to a search problem in a lattice of clauses organized by the (efficient to compute) θ -subsumption operator [2]. Several algorithms and implementations are based on this approach, and, in the work described here, we use the Aleph implementation of the Progol algorithm [3].

In Progol systems background predicates provided to the system define prior world knowledge and are the building blocks for the constructed hypothesis; that is to say, they define the literals that will be used in the body of the hypothesised clauses. The search for clauses within the search lattice defined by the background predicates is driven by the refinement operator, but also by semantic prior knowledge that encodes restrictions on the types of the variables of a clause and on the types that each predicate may accept.

B. Inducing Semantic Knowledge

DL statements formally describe *concepts* (classes, unary predicates) using complex class expressions involving the elementary class descriptions supported by each given DL. As ILP is restricted to the domain of definite Horn clause programs, which is a different first-order fragment than that of DLs, applying ILP to learn DL descriptions is not straightforward.

One approach explored is to embed DL concepts in the antecedents of Horn clauses, so that background knowledge expressed in DLs can be utilized. Rouveirol & Ventos [4]

and Lisi & Malerba [5] propose such hybrid Horn-DL languages as well as the corresponding learning methodologies. Although successful, such approaches are focused towards utilizing DL descriptions as background when constructing Horn clauses, rather than constructing DL descriptions.

It is then expected that in order to construct concept descriptions one needs to modify the ILP refinement operator so that a space of DL hypotheses is explored instead of a Horn space. Badea & Nienhuys-Cheng [6] propose such a top-down refinement operator. Although it addresses the over-specificity of the clauses learnt by previous bottom-up approaches [7], it is restricted to a simple DL without negation, numerical quantification, and relation composition.

Lehmann & Hitzler [8] and Ianone et al. [9] address negation, but do not learn all descriptions possible in *SHOIN* and *SROIQ*, the DLs that respectively cover the widely used semantic web formalisms OWL and OWL 2.

C. Learning Uncertain Knowledge

Under *many-valued semantics* an individual's membership in a set gets associated with one of many (instead of two) possible values, typically within the real unit interval. The logical connectives are, accordingly, interpreted by numerical operators.

Previous approaches to many-valued ILP typically use thresholds to create crisp representations of many-valued logics and then apply crisp DL-learning methodologies. This approach has been proven to work well, but is based on the assumption that the domain can be quantized. Bobillo et al. [10], for instance, successfully applied this approach to learning many-valued description under Gödel (min-max) semantics.

What should be noted, however, is that mapping to an equivalent finite and crisp task relies on a property that is specific to the min-max semantics, namely that new numbers (not encountered in the base data) are never introduced by the many-valued algebra. By contrast, in the work described here we propose a methodology that does not make such a strong assumption about properties inherent to a particular semantics, but allows a variety of t-norms to be used.

III. CASTING DL INFERENCE IN PROLOG

Our methodology is based on the *propositionalization* of very expressive DL constructs so that they can be expressed within Logic Programming and used as background by ILP systems, rather than adapting ILP systems to cover DL constructs that fall outside Logic Programming (LP).

This is achieved by using Prolog meta-predicates to define expressive DL constructs, not expressible in a pure LP setting, but retaining ILP learnability results by restricting access to meta-predicates in such a way that no fundamental assumption about the ILP setting is violated.

We shall first briefly present the constructs required to fully cover *SROIQ* (and thus OWL 2) concept descriptions, then proceed to their many-valued extension, and finally propose our method of casting the latter in a Logic Programming framework.

TABLE I
SET-THEORETIC SEMANTICS OF DL LANGUAGES. C AND D ARE CONCEPT NAMES, R IS A RELATION NAME, n IS A NON-NEGATIVE INTEGER CONSTANT, \cdot^I IS THE INTERPRETATION FUNCTION, Δ^I IS THE DOMAIN OF INTERPRETATION, AND $|\cdot|$ IS SET CARDINALITY.

DL	INTERPRETATION
\top	Δ^I
\perp	\emptyset
$\neg C$	$\Delta^I \setminus C^I$
$C \sqcap D$	$C^I \cap D^I$
$C \sqcup D$	$C^I \cup D^I$
$\exists R.C$	$\{x \mid \exists y. (x, y) \in R^I \wedge y \in C^I\}$
$\exists R.\text{Self}$	$\{x \mid (x, x) \in R^I\}$
$\forall R.C$	$\{x \mid \forall y. (x, y) \in R^I \rightarrow y \in C^I\}$
$\geq n R.C$	$\{x \mid \{y \mid (x, y) \in R^I \wedge y \in C^I\} \geq n\}$
$\leq n R.C$	$\{x \mid \{y \mid (x, y) \in R^I \wedge y \in C^I\} \leq n\}$

A. The DL Formalism

DL statements involve *descriptions* of classes. Descriptions are complex class expressions that may be connected with class names via either implication or equivalence. Table I shows the elementary class descriptions of *SROIQ*, one of the most expressive DLs proposed [11]. These descriptions can be recursively combined to build complex class descriptions, which are used to express subsumption or equivalence axioms like the following:

$$\begin{aligned} \text{Train} \sqcap \geq 5 \text{ hasCar. (Open} \sqcap \text{Car)} &\sqsubseteq \text{LongOpenTrain} \\ \text{Train} \sqcap \forall \text{hasCar} \circ \text{hasLoad.Square} &\equiv \text{SquareLoadTrain} \end{aligned}$$

meaning that trains are in the *LongOpenTrain* class *if* they are pulling 5 or more open cars, and that trains are in the *SquareLoadTrain* class *if and only if* they are pulling cars that contain load of type *Square*. Universal quantification descriptions are called *value* restrictions, and numerical quantification descriptions are called *at-most* and *at-least* number restrictions.

There is also limited support for relation descriptions and axioms in DLs:

- declaring (pairs of) relations as being transitive, reflexive, irreflexive, symmetric, antisymmetric, or disjoint;
- using relation descriptions involving negation, inversion, and composition of more elementary relations;
- stating relation subsumption axioms; and
- referring to the universal relation.

DL reasoners are deductive inference engines that perform logical inference over ontologies expressed as DL knowledge bases and deduce implicit knowledge from what is explicitly stated in the knowledge base.

B. Many-valued Description Logics

In order to be able to capture domains where facts and knowledge might be vague or imprecise, the binary (true/false) semantics presented above are extended to *many-valued* semantics where an individual's membership in a set gets associated with one of many (instead of two) possible values, typically within the real unit interval.

The logical connectives are, accordingly, interpreted by numerical operators. In the work described here we use the Łukasiewicz t-norm and residuum [12], yielding the following valuations to logical formulae:

$$\begin{aligned}\deg(C \sqcap D) &= \max(0, \deg(C) + \deg(D) - 1) \\ \deg(C \sqsubseteq D) &= \min(1, 1 - \deg(C) + \deg(D))\end{aligned}$$

where $\deg(\cdot)$ is the valuation of a clause. The rest of the logical connectives are, then, defined so that they verify De Morgan’s Laws:

$$\begin{aligned}\deg(C \sqcup D) &= \min(1, \deg(C) + \deg(D)) \\ \deg(\neg C) &= 1 - \deg(C) \\ \deg(C \equiv D) &= 1 - \text{abs}(\deg(C) - \deg(D))\end{aligned}$$

The disjunction operator is also called the *t-conorm*.

Following Straccia [13], we interpret quantifiers using the Gödel-Dummett t-norm and t-conorm [14], also called *weak* conjunction and disjunction, instead of Łukasiewicz norms:

$$\begin{aligned}\deg(C \sqcap D) &= \min(\deg(C), \deg(D)) \\ \deg(C \sqcup D) &= \max(\deg(C), \deg(D))\end{aligned}$$

C. Representation and Basic Constructs

In order to cover DL expressivity lying beyond the strict logic programming framework, we make use of the full Prolog language (including meta-predicates) within the definitions of these predicates. This is not to say that the ILP algorithm is given access to meta-predicates for constructing clauses, but that meta-predicates (such as `findall/3`) are used to implement first-order predicates that express concepts that are not expressible in the logic programming framework of the pure Prolog language.

Instances in a class are represented as an (`<instance-name>`, `<degree>`) pair, denoting the degree of the instance’s belonging to the class. We shall call such pairs *m-terms* and lists of m-terms *nominal classes*. Nominal classes might be represented as (ordered) Prolog lists, but are treated as unordered sets of m-terms. So, for example:

```
[(train1, 0.9), (train2, 0.3)]
```

is such a class containing two instances, `train1` at a degree of 0.9 and `train2` at 0.3.

Class membership assertions are made through the `add_to_concept/2` predicate, which expects as arguments a class name and a nominal class, and adds the later to the extension of the former. Relation assertions are made through the `add_to_relation/3` predicate, which expects as arguments a relation name `R`, an instance name `I`, and a nominal class. This predicate records the assertions that the relation `R` holds between `I` and all instances in the m-terms of the nominal class, at the degree given in each m-term.

Classes are placed in a subsumption hierarchy through the `declare_concept/2` predicate, which accepts as arguments two class names and asserts that the first is subsumed by the second. All classes are subsumed by the built-in `thing` class. Relations are placed in a subsumption hierarchy, and can also have their range and domain restricted

TABLE II
PREDICATES IMPLEMENTING THE CONJUNCTION OF A SET OF INSTANCES WITH A DL CLASS. THE FIRST ARGUMENT IS A SET, THE MIDDLE ARGUMENTS ARE THE PARAMETERS NECESSARY FOR SPECIFYING THE DL CLASS, AND THE LAST ARGUMENT IS THE SET RESULTING FROM THEIR CONJUNCTION. THE SECOND COLUMN SHOWS THE CLASS PARAMETERS AND THE THIRD COLUMN THE CORRESPONDING DL CLASS. *C* IS A CLASS NAME, *R* A RELATION NAME, AND *n* A NON-NEGATIVE INTEGER.

Predicate	Parameters	DL Construct
<code>concept_select/3</code>	<i>C</i>	$\sqcap C$
<code>forall_select/4</code>	<i>R, C</i>	$\sqcap \forall R.C$
<code>atleast_select/5</code>	<i>n, R, C</i>	$\sqcap \geq nR.C$
<code>atmost_select/5</code>	<i>n, R, C</i>	$\sqcap \leq nR.C$
<code>self_select/3</code>	<i>R</i>	$\sqcap \exists R.\text{Self}$

by the `declare_relation/4` predicate, which accepts as arguments two relation names and two class names: the first relation is subsumed by the second, and has the first class as domain and the second as range. The built-in relation name `unirel` subsumes all relations.

Logical expressions can be built by conjoining nominal classes with named classes as well as with the DL class description constructs such as $\forall R.C$, $\exists R.C$ and so on. In our system, the predicates shown in Table II are implemented, which are sufficient for expressing all class descriptions in *SRIOQ*. Naturally, these predicates cannot be defined in pure logic programming terms, and Prolog meta-predicates, like `findall/3`, are used to implement the semantics of DL quantifiers.

More specifically, the predicate `concept_select (+In, +C, -Out)` succeeds by binding its third argument to an nominal class, such that the instance names are the same as in the nominal class `In`, but the degrees are calculated as follows: the membership degree of each instance in class `C` is calculated, and then the t-norm is applied to this degree and the degree in `In`.

When calculating an instance’s membership degree in `C`, the class hierarchy is taken into account, and members of subclasses of `C` inherit the same degree of membership in `C`. This is because inclusion axioms, e.g. $D \sqsubseteq C$, cannot be ‘weighted’ and are implicitly asserted at a degree of 1.0. Given this, solving the residuum formula for $\deg(D)$ yields $\deg(D) \geq \deg(C)$, whereas solving the equivalence-valuation formula yields $\deg(D) = \deg(C)$. This is interpreted (and accordingly implemented) as follows: when calculating an instance’s membership in `C`, all (transitive) subclasses `Di` of `C` are identified and the membership of the instance in each is checked. If only one of `Di` includes the instance, this degree is also taken to be the degree for `C`. If more, then the t-conorm is used to aggregate these values in the combined membership degree.

The rest of the predicates, `forall_select/4`, `atleast_select/5`, `atmost_select/5`, and `self_select/3`, are similarly implemented, taking into account the relation hierarchy, as well as the class hierarchy, when retrieving members. Existential quantification is not directly

provided, as it is a special case of `atleast_select/5`.

Using this mechanism, conjunctive DL expressions can be expressed by ‘threading’ In and Out variables. Complex class expressions in quantification constructs have to be named and separately defined, like in the following example:

```
concept_select(A, 'RoundCar', B) :-
    concept_select(A, 'Round', C),
    concept_select(C, 'Car', B),

concept_select(A, 'LongRoundTrain', B) :-
    concept_select(A, 'Train', C),
    atleast_select(C, 5, hasCar,
                  'RoundCar', B).
```

to mean:

$$\text{Train} \sqcap \geq 5 \text{ hasCar. (Round} \sqcap \text{Car)} \sqsubseteq \text{LongRoundTrain}$$

D. Quantified Expressions

As already mentioned, quantification is interpreted by weak conjunction and disjunction, regardless of the norms used to interpret the logical connectives.

More specifically, if ϕ is a first-order formula, the degree of $\forall x.\phi x$ is:

$$\deg(\forall x.\phi x) = \min_{x \in \Delta} \deg(\phi x)$$

which is straightforwardly implemented in `forall_select/4` by using the `findall/3` meta-predicate and then searching in the results for the lowest membership degree.

Similarly, the degree of $\exists x.\phi x$ is:

$$\deg(\exists x.\phi x) = \max_{x \in \Delta} \deg(\phi x)$$

and, since at-least restrictions can be expressed as:

$$\exists y_1 \exists y_2 \dots \exists y_n. Rxy \wedge \bigwedge_{1 \leq i \leq n} Cy_i \wedge \bigwedge_{\substack{1 \leq i \leq n-1 \\ i < j \leq n}} y_i \neq y_j$$

the implementation of `atleast_select/5` proceeds as follows to evaluate the expression $\geq nR.C$:

- 1) retrieves all instance pairs (a_i, b_i) that have relation R at non-zero degrees r_i
- 2) calculates the degrees c_i at which instances b_i are members of concept C
- 3) applies the Łukasiewicz t-norm to all (r_i, c_i) pairs to create a new list of (b_i, t_i) pairs
- 4) removes all duplicates among b_i , retaining the (b_i, t_i) pairs with the highest t_i value
- 5) the result is the minimum of the n highest values in the list of t_i values, since n elements are weak-conjoined, after being existentially selected.

At-most restrictions are implemented in `atmost_select/5` in a similar fashion.

E. Complex Expressions

In the previous section we have described a mechanism for expressing conjunctive class descriptions, without negation, equivalence, and disjunction. Furthermore, there was no mention of relation composition.

Complementization is expressed by the `compl/1` operator, which creates a new class where membership degree is calculated by applying the negation norm to the membership degree in the original class. This mechanism lies in the implementation of the `_select` predicates of Table II. In this manner, when an expression like the following:

```
concept_select(A, 'Train', C),
atleast_select(C, 5, hasCar,
              compl('RoundCar'), B).
```

is evaluated, the nominal class that is built by `RoundCar` is then applied to the negation norm, which creates a new nominal class with the same instances and new degrees.

Disjunction is handled by multiple Prolog clauses, each providing alternative ways to prove class membership, and combined according to the t-conorm. For example, given the clauses:

```
concept_select(A, 'LongOrRoundTrain', B) :-
    concept_select(A, 'Train', C),
    atleast_select(C, 5, hasCar, 'Car', B).

concept_select(A, 'LongOrRoundTrain', B) :-
    concept_select(A, 'Train', C),
    forall_select(C, hasCar, 'Round', B).
```

an instance that satisfies both clauses will be considered as being a member of `LongOrRoundTrain` to a degree that is derived by using `findall/3` to collect all alternative membership degrees, and then applying the t-conorm to combine them into the degree of the disjunction. In this manner, the clauses above implement the semantics of:

$$\text{Train} \sqcap (\geq 5 \text{ hasCar.Car} \sqcup \forall \text{hasCar.Round}) \sqsubseteq \text{LongOrRoundTrain}$$

after applying all the necessary transformations to bring it into clausal form. In our implementation, we are freely applying De Morgan’s Laws to perform these transformations, and hence it is necessary that they are verified by the t-norm used. This is, indeed, the case with the Łukasiewicz t-norm used in the work described here. Supporting norm systems where De Morgan’s Laws are not tautological is also possible, but the computation is more complicated as the norms will have to be applied at each transformation step.

Finally, chains of relation-composition expressions are supported by backtracking through possible ‘connecting’ instances, looking for a series of relation members that satisfies the domain and range restrictions of the relations in the chains. Relation chains are represented as lists of relation names, so that a Prolog expression such as:

```
atleast_select(A, 3,
              [hasCar, hasLoad], 'Round', B)
```

retrieves instances that satisfy the description:

$$\geq 3 \text{ hasCar} \circ \text{hasLoad}.\text{Round}$$

IV. EXPERIMENTS AND RESULTS

We demonstrate our approach on a fuzzy variation of the famous trainspotting machine-learning problem and on the Iris dataset, using the ALEPH [3] implementation of the PROGOL algorithm and our implementation of the DL inference described above.

The learning task is to construct the two-argument predicate that relates a ground input nominal class with an output variable. The examples are instances of this predicate where all class members participate at a degree of 1.0 in the input and at varying degrees in the output. Background and semantic bias restricts the search space to `_select` predicates, with typed predicate arguments (concept names, relation names, and nominal class variables), and calling modes enforcing the variable threading described in Section III.

Furthermore, syntactic bias is used to complement the threaded-variable semantic constraint: although threading is already semantically enforced, it is not possible to specify by variable semantics that all body literals in a clause will form a *single* thread. This is enforced by a syntactic checker that drops hypotheses where the output variable of each literal is not the input variable of exactly one other literal—taking care to correctly handle the first and last body literals.

A. Trainspotting

We solve a many-valued variation of the eastbound-trains problem [15], defining five instances of the `Train` class and a number of `Car` and `Load` instances. Membership degrees in the `Train` class denote the degree of similarity to the ‘perfect train engine’ (Train 4 in Figure 1). All carriages and loads belong in their respective classes at the same (very high) degree.

`Car` instances are also instances of the `Closed` class. For example, the last car of Train 1 has a very high degree, as opposed to the first car of Train 4. `Load` instances are also instances of the `Triangle` and `Square` classes at varying degrees, ranging from almost perfect triangle/no square to almost perfect square/no triangle (compare the two loads of Train 5). Some loads might have a non-zero degree of membership in both classes, like the load falling off the last car of Train 2.

Each `Train` instance may have multiple `hasCar` relations with `Car` instances and each `Car` instance may have multiple `hasLoad` relations with `Load` instances. Relations also hold at varying degrees, showing the strength of the association; in Figure 1, dotted lines denote a `hasCar` relationship of degree less than 1, with less solid lines representing smaller degrees. The degree of the `hasLoad` relations is depicted by the distance of the load from the floor of the car, ranging from being firmly fixed to the floor (degree of 1) to falling off the car.

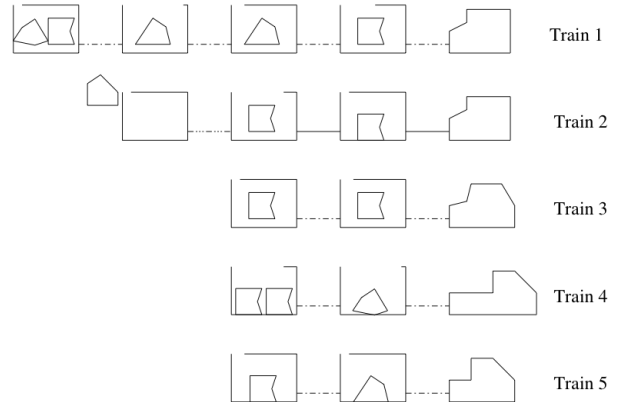


Fig. 1. Schematic depiction of the eastbound-trains domain.

Trains t_2 and t_3 constitute the positive datapoints, and trains t_1, t_4 and t_5 the negative ones.

The training set is represented as ground instances of the `eastbound/2` predicate. For example,

```
eastbound( [(t2, 1.0)], [(t2, 0.8)] ).
```

is a positive example. Notice that the `In` and `Out` set are singletons. The degree 0.8 represents lowest valuation for which the example is to be considered covered. Note that, in order to be of any consequence, positive examples must have a threshold degree close to 1.0 and negative ones a threshold degree close to 0.0. For instance, a positive example such as:

```
eastbound( [(t3, 1.0)], [(t3, 0.3)] ).
```

will not influence the search as much as t_2 shown above, since its low coverage requirements might get satisfied by clauses that would not cover it at a higher threshold.

After visiting 50678 nodes during reduction, the following clause describing eastbound trains is discovered:

```
eastbound(A, B) :-
    atleast_select(A, [hasCar], thing,
                  3, C),
    forall_select(C, [hasCar, hasLoad],
                 'SquareLoad', B).
```

which corresponds to the following DL description:

$$\leq 3 \text{ hasCar} \sqcap \forall \text{hasCar} \circ \text{hasLoad}.\text{SquareLoad} \sqsubseteq \text{Eastbound}$$

Indeed, Trains 2 and 3 satisfy `eastbound/2` at a higher degree than the threshold. Intuitively, the trains in the eastbound class have at most three cars, and carry square loads. As one can see in Figure 1, Train 2 is only loosely connected with the third car, which contains a load that looks more like a triangle load than a square one. This is not a counterexample to the learned rule, because the degree that relates the train with the triangle load is low enough to consider Train 2 as a train with only square loads. Consequently, Train 2 is a member of eastbound in a higher

TABLE III
RECALL, PRECISION, AND $F_{\beta=1}$ SCORE FOR IRIS CATEGORIZATION.

Clause	Recall	Precision	F_{β} -score
I. setosa	100%	100%	100%
I. virginica	56%	100%	72%
I. versicolour	18%	82%	30%

degree than the threshold. Furthermore, none of the negative examples satisfies the predicate to a high enough degree to be a counterexample to the rules' validity.

B. Iris Setosa

A classic machine-learning data set is Iris, originally compiled by Fisher [16] and now available at the UCI repository [17]. The data set contains three botanical classes (I. setosa, I. versicolour, and I. virginica) of 50 instances each, where each class refers to a type of iris plant. Four numerical features are provided (sepal length, sepal width, petal length, and petal width); using these features I. setosa is linearly separable from the other two, which are not linearly separable from each other.

We used this data to prepare three experiments, each targeting the description of one of the iris classes and using as background knowledge the four many-valued classes that correspond to the four features in the dataset (LongSepals, WidePetals, etc.) All instances in the dataset are asserted as members of all background classes at a degree derived by uniformly distributing the raw measurements in the $[0, 1]$ range.

Aleph constructs the following axioms:

$$\begin{aligned}
 \neg \text{LongSepals} \sqcap \neg \text{LongPetals} &\sqsubseteq \text{ISetosa} \\
 \text{LongPetals} \sqcap \text{LongPetals} &\sqsubseteq \text{IVirginica} \\
 \neg \text{WideSepals} \sqcap \neg \text{WideSepals} \sqcap \\
 \neg \text{WideSepals} &\sqsubseteq \text{IVersicolour}
 \end{aligned}$$

Note that the duplicate literal in the I. virginica clause is meaningful under Łukasiewicz semantics, and means 'with very long petals'. In other words, it takes a bigger raw measurement for petal length in order to achieve the same degree of membership than a single occurrence of the literal would.

We evaluate the accuracy of the constructed axioms by selecting for each instance the Iris class at which it belongs at the highest degree, and observe that our method constructed a perfect definition for I. setosa but, as expected, cannot perfectly distinguish I. versicolour from I. virginica; see Table III for precision, recall, and F-score figures.

V. CONCLUSIONS

Despite the considerable research carried out towards adapting ILP to Description logics, current approaches are restricted to the less expressive members of the DL family. We propose a novel method where we approach machine learning of DLs from the opposite direction, by developing a means of expressing class descriptions *within the Logic programming framework*, allowing for the direct application

of well-trying and highly-optimized ILP systems to the task of synthesising class descriptions.

The resulting methodology is, effectively, an instance-based DL reasoning service implemented as a transformation of DL expressions to Prolog clauses that implement the semantics of DL operators. Such clauses use Prolog meta-predicates to implement the aspects of DL that fall outside LP expressivity, but do so in a way that does not fall outside the application domain of example-setting ILP systems.

It is obvious that the resulting reasoning service is poor by today's state-of-the-art: it fails to observe open-world semantics; there is no equivalence; and cardinality restrictions refer to the explicitly asserted relations only, disregarding consistent but not explicitly made statements. What should be stressed, however, is that this is not important: the inference is sound (although not complete), and, since we operate in the example setting, concept descriptions are validated over explicit data only anyway. In other words, open-world reasoning is not needed to validate the hypothesised clauses over the training examples.

We demonstrate our system on classic and well-studied machine learning problems: the smaller, artificially constructed, train-spotting problem and the larger, naturally observed Iris dataset. The train-spotting problem is rich in relations and relation chains, demonstrating the method's adequacy in constructing complex expressions. The Iris dataset is structurally simpler, but demonstrates the method's ability to handle numerical valuations beyond thresholding and the min-max algebra.

Our future research includes investigating a principled way of extracting more ILP bias from DL models, in order to increase the efficiency of the ILP run. Furthermore, we are planning to re-design the clausal-form transformations so that any norm system can be applied and not only those that verify De Morgan's Laws.

Another technical detail lies in the preparation of the ILP runs: in the proof-of-concept experiments presented here we semi-automatically prepared our ILP runs using custom-made scripts. We are in the process of building upon the TRANSONTO knowledge transformation system¹ to implement a library that automatically transforms OWL data into our Prolog formulation as well as Aleph output back into OWL, in order to allow integration into ontology maintenance systems.

ACKNOWLEDGMENTS

S. Konstantopoulos wishes to acknowledge the support of FP7-ICT PRONTO, funded by the European Commission. See <http://www.ict-pronto.org> for more details.

A. Charalambidis wishes to acknowledge the partial support of DELTIO, funded by the Greek General Secretariat of Research & Technology.

¹See <http://transonto.sourceforge.net/>

REFERENCES

- [1] S. Konstantopoulos, R. Camacho, N. A. Fonseca, and V. Santos Costa, "Induction as a search," in *Artificial Intelligence for Advanced Problem Solving Techniques*, D. Vrakas and I. Vlahavas, Eds. Hershey, PA, USA: IGI Global, Jan. 2008, ch. VII, pp. 158–205.
- [2] S. Muggleton, "Inverse entailment and Progol," *New Generation Computing*, vol. 13, pp. 245–286, 1995.
- [3] A. Srinivasan, *The Aleph Manual*, Last update: June 30, 2004. [Online]. Available: <http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/>
- [4] C. Rouveirol and V. Ventos, "Towards learning in CARIN- \mathcal{ALN} ," in *Proceedings of the 10th International Workshop on Inductive Logic Programming (ILP 2000)*, ser. Lecture Notes in Artificial Intelligence, J. Cussens and A. M. Frisch, Eds., vol. 1866. Berlin/Heidelberg: Springer Verlag, 2000, pp. 191–208.
- [5] F. A. Lisi and D. Malerba, "Bridging the gap between horn clausal logic and description logics in inductive learning," in *Advances in Artificial Intelligence: Proc. of 8th Congress of the Italian Association for Artificial Intelligence, Pisa, Italy, 23-26 Sep, 2003*, ser. Lecture Notes in Computer Science, vol. 2829. Berlin/Heidelberg: Springer-Verlag, 2003, pp. 53–64.
- [6] L. Badea and S.-H. Nienhuys-Cheng, "A refinement operator for description logics," in *Proceedings of the 10th International Workshop on Inductive Logic Programming (ILP 2000)*, ser. Lecture Notes in Artificial Intelligence, J. Cussens and A. M. Frisch, Eds., vol. 1866. Berlin/Heidelberg: Springer Verlag, 2000, pp. 40–59.
- [7] 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, pp. 121–133.
- [8] J. Lehmann and P. Hitzler, "A refinement operator based learning algorithm for the \mathcal{ALC} description logic," in *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP 2007)*, Corvallis, Oregon, U.S.A., Jun, 19–21, ser. Lecture Notes in Computer Science, H. Blockeel, J. Ramon, J. Shavlik, and P. Tadepalli, Eds., vol. 4894. Berlin/Heidelberg: Springer Verlag, Feb. 2008, pp. 147–160.
- [9] L. Iannone, I. Palmisano, and N. Fanizzi, "An algorithm based on counterfactuals for concept learning in the semantic web," *Journal of Applied Intelligence*, vol. 26, no. 2, pp. 139–159, Apr. 2007.
- [10] F. Bobillo, M. Delgado, and J. Gómez-Romero, "Optimizing the crisp representation of the fuzzy description logic \mathcal{SRQIQ} ," in *3rd Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2007)*, Busan (South Korea), November 2007., 2007.
- [11] I. Horrocks, O. Kutz, and U. Sattler, "The even more irresistible \mathcal{SRQIQ} ," in *Proceedings of the 10th Int. Conference on Principles of Knowledge Representation and Reasoning (KR 2006)*, 2006.
- [12] J. Lukasiewicz and A. Tarski, "Untersuchungen uber den Aussagenkalkül," *Comptes Rendus des Séances de la Société des Sciences et des Lettres de Varsovie, Classe III*, vol. 23, 1930, reprinted in translation in L. Borowski (ed.), *Jan Lukasiewicz, Selected Writings*, North-Holland, 1970.
- [13] U. Straccia, "Description Logics with fuzzy concrete domains," in *21st Conference on Uncertainty in Artificial Intelligence (UAI05)*, Edinburgh, 2005, F. Bachus and T. Jaakkola, Eds., 2005, pp. 559–567.
- [14] M. Dummett, "A propositional calculus with denumerable matrix," *Journal of Symbolic Logic*, vol. 24, pp. 97–106, 1959.
- [15] J. Larson and R. S. Michalski, "Inductive inference of VL decision rules," *ACM SIGART Bulletin*, vol. 63, pp. 38–44, Jun. 1977.
- [16] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annual Eugenics*, vol. 7, no. II, pp. 179–188, 1936, reprinted in *Contributions to Mathematical Statistics* (John Wiley, NY, 1950).
- [17] A. Asuncion and D. J. Newman, "UCI machine learning repository," University of California, Irvine, School of Information and Computer Sciences, 2007. [Online]. Available: <http://archive.ics.uci.edu/ml/>
- [18] J. Cussens and A. M. Frisch, Eds., *Proceedings of the 10th International Workshop on Inductive Logic Programming (ILP 2000)*, ser. Lecture Notes in Artificial Intelligence, vol. 1866. Berlin/Heidelberg: Springer Verlag, 2000.