Downward Refinement in the ALN Description Logic.

Conference Paper · January 2004

DOI: 10.1109/ICHIS.2004.39 · Source: DBLP

5 authors, including:

Nicola Fanizzi
Università degli Studi di Bari Aldo Moro
274 PUBLICATIONS **2,565** CITATIONS
SEE PROFILE

Stefano Ferilli
Università degli Studi di Bari Aldo Moro
367 PUBLICATIONS **2,065** CITATIONS
SEE PROFILE

Luigi Iannone
Università degli Studi di Salerno
61 PUBLICATIONS **812** CITATIONS
SEE PROFILE

Ignazio Palmisano
The University of Manchester
63 PUBLICATIONS **569** CITATIONS
SEE PROFILE

Downward Refinement in the $\mathcal{ALN}$ Description Logic

Nicola Fanizzi, Stefano Ferilli, Luigi Iannone, Ignazio Palmisano and Giovanni Semeraro
Dipartimento di Informatica, Università degli Studi di Bari, Via Orabona 4, 70125, Bari, Italy
lastname@di.uniba.it

Abstract

We focus on the problem of specialization in a Description Logics (DL) representation, specifically the $\mathcal{ALN}$ language. Standard approaches to learning in these representations are based on bottom-up algorithms that employ the lcs *operator, which, in turn, produces overly specific (*overfitting*) and still redundant concept definitions. In the dual (top-down) perspective, this issue can be tackled by means of an ILP downward operator. Indeed, using a mapping from DL descriptions onto a clausal representation, we define a specialization operator computing maximal specializations of a concept description on the grounds of the available positive and negative examples.*

Keywords: *Description Logics, Knowledge Refinement*

1. Introduction and Motivation

The markup languages that are employed in the Semantic Web [4] have Description Logics (henceforth DLs) as their theoretical foundation [1]. We examine the problem of the induction and refinement of definitions for the concepts and their properties starting from assertions made on individuals available in a knowledge base. This case requires to deal with different languages because DLs represent a peculiar fragment of FOL, with different expressive power w.r.t. clausal logics [5]. In *Inductive Logic Programming* (ILP), several extensions of the relational frameworks have been proposed, spanning from more expressive languages, such as the *prenex conjunctive normal forms* [17] to hybrid representations that mix DLs and LP, such as $\mathcal{AL}$-log [8], CARIN-$\mathcal{ALN}$ [18] and *DLP* [14]. Since DL representations can add numeric restrictions, the semantics of the ILP component must allow for counting. Specific interpretations like the *Unique substitution semantics* [13] or the *Object Identity semantics* [10] are to be taken into account. DL representations can be compared to the graph representations of early works in the field [12]. In order to cope with the problem complexity, former methods, e.g. the *least common subsumer* (*lcs*) [6], were based on a heuristic search and the algorithms tend to induce overly specific concept definitions(*overfitting*). Other approaches have shown that also a top-down search of conceptual definitions in a specific DL is feasible [3], yet the refinement is less operational: it is intended to show the properties of the related search space. A flexible method that can be used incrementally is needed. In fact, whenever new assertions in the A-box make it inconsistent w.r.t. the T-box, it can be difficult to refine the concept definitions accordingly. Even using the above mentioned *lcs* (which depends on the underlying DL), the whole work has to be repeated each time inconsistency is detected in the knowledge base. In this work, we present refinement operators for $\mathcal{ALN}$ concept descriptions that can be exploited also for refining terminological components in the hybrid representations. Moreover we show that the operators are complete. Furthermore, we intend to involve information coming from examples directly in the induction of candidate refinements. Exploiting a transformation proposed by Kietz [14] from a DL into a clausal representation, we are able to solve the downward refinement problem with an ILP algorithm which can be easily carried out by using existing systems.

After the next Section 2 with the preliminaries of the representation languages involved, in Sect. 3 the search space is presented and the refinement operators are defined. Then the DL learning problem is turned into an ILP one with a transformation recalled in Sect. 4. This allows for an ILP specialization method taking into account the examples during the refinement process (covered by Sect. 5). Finally, possible extensions are discussed in Sect. 6.

2. The $\mathcal{ALN}$ DL Representation Language

In this section we define syntax and semantics for the reference representation $\mathcal{ALN}$ adopted in the paper. This language was chosen because of the tractability of the related reasoning services [7] and of its applicability to hybrid representations. Moreover, it has been shown how they can be transformed [14] into a (constraint) logic programming representation. This is discussed in Sect. 4.

In a DL language, primitive *concepts* $N_C = \{C, D, \ldots\}$ are interpreted as subsets of a certain domain of objects and

Constructor Name	Syntax	Semantics
top concept	$\top$	Δ
bottom concept	$\bot$	$\emptyset$
primitive negation	$\neg P$	$\Delta \setminus P^{\mathcal{I}}$
concept conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
value restriction	$\forall R.C$	$\{x \in \Delta \mid \forall y \, (x,y) \in R^{\mathcal{I}} \to y \in C^{\mathcal{I}}\}$
at_most restriction	$\leq n.R$	$\{x \in \Delta \mid \, \lvert\{y \in \Delta \mid (x,y) \in R^{\mathcal{I}}\}\rvert \leq n\}$
at_least restriction	$\geq n.R$	$\{x \in \Delta \mid \, \lvert\{y \in \Delta \mid (x,y) \in R^{\mathcal{I}}\}\rvert \geq n\}$

Table 1. DL constructors and interpretation.

primitive *roles* $N_R = \{R, S, \ldots\}$ are interpreted as binary relations on such a domain. More complex concept descriptions are built using atomic concepts and primitive roles by means of the constructors presented in Table 1.

Their meaning is defined by an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the *domain* of the interpretation and the functor $\cdot^{\mathcal{I}}$ stands for the *interpretation function* mapping the intension of concept and role descriptions to their extension. A *knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ contains two components: a T-box $\mathcal{T}$ and an A-box $\mathcal{A}$. $\mathcal{T}$ is a set of concept definitions $C \equiv D$, meaning $C^{\mathcal{I}} = D^{\mathcal{I}}$, where C is the concept name and D is a description given in terms of the language constructors. Differently from ILP, each (non primitive) concept has a single definition. Moreover, we assume that the definitions are not cyclic. $\mathcal{A}$ contains extensional assertions on concepts and roles, e.g. $C(a)$ and $R(a,b)$, meaning, respectively, that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

Note that the concept description C are not limited to LP facts but can be more structured. For instance they could assert a universal property of the an individual: $(\forall R.C)(a)$ that is, a is necessarily related through the role R to individuals that are instances of concept C. The semantic notion of *subsumption* between DL concept descriptions can be given in terms of the interpretations defined above:

Definition 2.1 (subsumption) *Given two concept descriptions C and D in $\mathcal{T}$, C* subsumes *D iff for every interpretation $\mathcal{I}$ of $\mathcal{T}$ it holds that $C^{\mathcal{I}} \supseteq D^{\mathcal{I}}$. This is denoted by $C \sqsupseteq_{\mathcal{T}} D$, where $\mathcal{T}$ can be omitted when it is obvious. This induces also an equivalence relationship, denoted $C \equiv D$, that amounts to both $C \sqsupseteq D$ and $D \sqsupseteq C$.*

Note that the definition is merely semantic and independent from the particular DL language adopted. Many semantically equivalent (yet syntactically different) descriptions can be given for the same concept; they can be reduced to a canonical form by means of rewriting rules that preserve their equivalence, e.g. $\forall R.C_1 \sqcap \forall R.C_2 \equiv \forall R.(C_1 \sqcap C_2)$ (for issues related to normalization and simplification, see [1]).

The normal form employs the notation needed to access the different parts of concept descriptions (*sub-descriptions*):

- $\mathsf{prim}(C)$ denotes the set of all (negated) concept names occurring at the top level of C;
- $\mathsf{val}_R(C)$ denotes conjunction of concepts $C_1 \sqcap \cdots \sqcap C_n$ in the value restriction of role R, if any (otherwise $\mathsf{val}_R(C) = \top$);
- $\mathsf{min}_R(C) = \max\{n \in \mathbb{N} \mid C \sqsubseteq (\geq n.R)\}$ (always a finite number);
- $\mathsf{max}_R(C) = \min\{n \in \mathbb{N} \mid C \sqsubseteq (\leq n.R)\}$ (if unlimited then $\mathsf{max}_R(C) = \infty$).

Definition 2.2 ($\mathcal{ALN}$ normal form) *A concept description C is in $\mathcal{ALN}$ normal form iff $C = \top$ or $C = \bot$ or*

$$C = \prod_{P \in \mathsf{prim}(C)} P \sqcap \prod_{R \in N_R} (\forall R.C_R \sqcap {\geq} n.R \sqcap {\leq} m.R)$$

where: $C_R = \mathsf{val}_R(C)$, $n = \mathsf{min}_R(C)$ and $m = \mathsf{max}_R(C)$

The complexity of normalization is polynomial, as shown also in [14]. Hence subsumption checking is accordingly polynomial like $O(n \, log \, n)$. Subsumption between concept descriptions is a semantic relationship, thus a more syntactic relationship is needed. A language of moderate complexity like $\mathcal{ALN}$ allows also for a structural characterization of subsumption [1].

Proposition 2.1 (subsumption in $\mathcal{ALN}$) *Given two $\mathcal{ALN}$ concept descriptions C and D in normal form, it holds that $C \sqsupseteq D$ iff all the following relations hold between the sub-descriptions:*

- $\mathsf{prim}(C) \subseteq \mathsf{prim}(D)$
- $\forall R \in N_R\colon\ \mathsf{val}_R(C) \sqsupseteq \mathsf{val}_R(D)$
- $\mathsf{min}_R(C) \leq \mathsf{min}_R(D)$
- $\mathsf{max}_R(C) \geq \mathsf{max}_R(D)$

Another characterization of subsumption in $\mathcal{ALN}$ that gives a way to compute the *lcs* of two concept descriptions is based on tree structures that can be associated to the descriptions:

Definition 2.3 (description tree) *An $\mathcal{ALN}$* description tree *is a tree $\mathcal{G} = (V, E, v_0, l)$ with root v_0 where the empty label corresponds to $\top$ and:*

- *each edge in E is labelled with $\forall R$, $R \in N_R$*
- *each node $v \in V$ is labelled with a finite set: $l(v) \subseteq N_C \cup \{\geq n.R \mid n \in \mathbb{N},\ R \in N_R\} \cup \{\leq n.R \mid n \in \mathbb{N},\ R \in N_R\}$*

A concept description C in normal form corresponds to the tree $\mathcal{G}_C = (V, E, v_0, l)$ defined recursively on m number of value restrictions:

$(m = 0)$ $\quad \mathcal{G}_C = (\{v_0\}, \emptyset, v_0, l)$ *with* $l(v_0) = \mathsf{prim}(C) \cup \bigcup_{R \in N_R} ({\geq} n.R \sqcap {\leq} m.R)$

$(m > 0)$ *let $\mathcal{G}_i = (V_i, E_i, v_{i0}, l_i)$ be the description tree of $C_i, 1 \leq i \leq m$ where w.l.o.g. the V_i are pairwise disjoint and $v_0 \notin \bigcup_{1\leq i\leq m} V_i$. Then:*

- $V = \{v_0\} \cup \bigcup_{1\leq i\leq m} V_i$
- $E = \{v_0 R_i v_{i0} \mid 1 \leq i \leq m\} \cup \bigcup_{1\leq i\leq m} E_i$
- *if* $v = v_0$ *then* $l(v) = \mathsf{prim}(C) \cup \bigcup_{R\in N_R}(\geq n.R \sqcap \leq m.R)$
  otherwise $l(v) = l_i(v)$ *(being $v \in V_i$)*

This relationship can be exploited to define a notion of coverage of concept descriptions with respect to the assertions, which is useful also during the learning process when testing the induced descriptions. However, here the most important difference between a DL and clausal representation arises. While in the context of DL reasoning the *Open World Assumption* (OWA) is required, in ILP the *Closed World Assumption* (CWA) is generally adopted. Thus, a different notion of explanation becomes essential to test the candidate solutions of a learning problem. Finally, testing whether an individual is an instance of a concept can be difficult. This is solved by an inference service that computes the *most specific concept* description (*msc*) [15, 1]. However this is not possible in all the DL languages. When the A-boxes are cyclic, only an approximation of the *msc* can be computed [6, 2]. In this case, the descriptive semantics is not adequate and the greatest-fixpoint has to be assumed [15, 2, 1].

3. Refinement Operators for $\mathcal{ALN}$

In a DL context, the supervised learning problem can be formally defined as follows:

Definition 3.1 (learning problem)
Given: *a knowledge base $\mathcal{K} = \langle\mathcal{T}, \mathcal{A}\rangle$, a set of positive and negative assertions $\mathcal{A}_C = \mathcal{A}_C^+ \cup \mathcal{A}_C^-$ regarding the membership of some individuals to a concept C such that:*

$$\mathcal{T} \not\models \mathcal{A} \cup \mathcal{A}_C$$

Find *a T-box $\mathcal{T}' = (\mathcal{T} \setminus \{C \equiv D\}) \cup \{C \equiv D'\}$ such that:*

$$\mathcal{T}' \models \mathcal{A} \cup \mathcal{A}_C$$

This problem can be cast as a search in a space of candidate refinements. The definition of the *search space* depends on the ordering adopted over the concept descriptions. This notion induces a generalization model (a quasi-ordering) that gives a criterion for traversing the space of solutions by means of suitable operators.

Definition 3.2 (refinement operators) *Given a quasi-ordered set $(\mathcal{S}, \succeq)$, a* downward *(resp.* upward*)* refinement operator *ρ (resp. δ) is a mapping from $\mathcal{S}$ to $2^{\mathcal{S}}$, such that $D' \in \rho(D)$ implies $D \succeq D'$ (resp. $D' \in \delta(D)$ implies $D' \succeq D$).*

The closure *of the refinement operator τ for $C \in \mathcal{S}$ is defined: $\tau^*(C) = \bigcup_{n\geq 0} \tau^n(C)$, where $\tau^0(C) = \{C\}$ and $\tau^n(C) = \{D \in \mathcal{S} \mid \exists E \in \tau^{n-1}(C)\colon\ D \in \tau(E)\}$.*

Within an ordered space, refinement operators represent the theoretical key for the treatment of learning as decoupled into search and heuristics. They have been investigated for refining logic programs in ILP [16, 9]. Many properties have been defined in the literature on refinement operators (e.g. see [3]). The properties of the operators depend on the order; we adopt the one induced by the subsumption relationship. Now it is possible to define refinement operators that perform the basic operations for computing specialized (or generalized) concept descriptions w.r.t. the given input:

Definition 3.3 (downward refinement operator in $\mathcal{ALN}$)
A downward refinement operator ρ is defined as follows. Given an $\mathcal{ALN}$ concept description D:

- *$D \sqcap L \in \rho(D)$ where L is a (negated) primitive concept, a number or value restriction and no L' at the top level of D is such that $L \sqsupseteq L'$*
- *$D \sqcap \forall R.C_1 \in \rho(D \sqcap \forall R.C_2)$ with $C_1 \in \rho(C_2)$*
- *$D \sqcap \leq n.R \in \rho(D \sqcap \leq m.R)$ if $m > 0$ and $n = m - 1$*
- *$D \sqcap \geq n.R \in \rho(D \sqcap \geq m.R)$ if $n = m + 1$*

Definition 3.4 (upward refinement operator in $\mathcal{ALN}$)
An upward refinement operator δ is defined as follows. Given an $\mathcal{ALN}$ concept description D:

- *$D \in \delta(D \sqcap L)$ where L is a (negated) primitive concept, a number restriction or a value restriction*
- *$D \sqcap \forall R.C_1 \in \delta(D \sqcap \forall R.C_2)$ with $C_1 \in \delta(C_2)$*
- *$D \sqcap \leq n.R \in \delta(D \sqcap \leq m.R)$ if $n = m + 1$*
- *$D \sqcap \geq n.R \in \delta(D \sqcap \geq m.R)$ if $m > 0$ and $n = m - 1$*

Ideal refinement operators have been proved not to exists in spaces where (uncovered) infinite chains of elements can be constructed (see [16]). In the case of $(\mathcal{ALN}, \sqsupseteq)$, the following infinite chain can be considered: $E_0 = (\leq 1.R) \sqcap (\geq 1.R)$ and $E_{n+1} = \forall R.E_n$ with $n \in \mathbb{N}$. The completeness of the operators defined above can be proved by induction on concept tree depth. It relies on the characterization of subsumption recalled by Proposition 2.1. A similar result applies to the upward refinement case. Defining refinement operators with good properties allows to devise a generate and test algorithm whose complexity depend on the structure of the search space. Yet there is the information coming from the examples which is not exploited in the refinement process. In the following, we first convert the DL induction problem into an ILP one, then we give operators for performing specialization (and generalization) which are based also on the available assertions.

4. Description Logics and Clausal Logics

The relationships between DL and LP suggest how to bridge the gap between them. Especially for simpler languages of the DLs family like $\mathcal{ALN}$, a conversion across the representations is possible. Thus, learning techniques and systems devised in the ILP field can be exploited to solve the learning problem in both the frameworks and also for hybrid languages mentioned in Sect.1. Indeed, the study of the learnability of some hybrid languages like DLP, based on both DL and LP, has required the transformation of $\mathcal{ALN}$ descriptions into clausal logic [14]. A similar transformation from DL descriptions into Horn clauses is briefly recalled in the following:

Definition 4.1 (transformation) *Given a concept description in normal form D, it can be transformed into a Horn clause by applying the following rules:*

- $\Phi(D, X_1) = h(X_1) \cup \Phi(\mathsf{prim}(D), X_1) \cup \cup \bigcup_{R\in N_R} \Phi(\forall R.C_R \sqcap \geq n.R \sqcap \sqcap \leq m.R, X_1)$
- $\Phi(\mathsf{prim}(D_i), X_k) = \bigcup_{P\in\mathsf{prim}(D_i)} \Phi(P, X_k)$
- $\Phi(P, X_k) = cp_P(X_k)$ *and* $\Phi(\neg P, X_k) = cn_P(X_k)$ *for all primitive concepts at level* k
- $\Phi(\forall R.C_R \sqcap \geq n.R \sqcap \leq m.R, X_k) = = \{rr_R(X_k, X_{k+1}, n, m), C_R(X_{k+1})\}$ *where, if* $C_R = \top$ *then* $C_R(X_{k+1})$ *is omitted and the absence of number restrictions is treated by using the special values* $n = 0$ *and/or* $m = \infty$

The standard LP semantics must be augmented to accommodate also the new terms representing intervals of integers. As shown in [14], this can be achieved by using Constraint ILP techniques [19]. Therefore, an interval $[a, b]$ is more general than another $[c, d]$ iff $a \geq c$ and $b \leq d$. Moreover, the least generalization of two such terms is the smallest interval that includes both. The case of $\bot(X)$ is special since it has to be treated as a literal that is subsumed by any other one. This is also taken into account in computing the *lgg* of such literals w.r.t. the others.

5. ILP to Refine $\mathcal{ALN}$ Concept Descriptions

In this section we show that, exploiting the transformation function, it is possible to employ ILP techniques to solve the learning problem for $\mathcal{ALN}$.

The problem of generalization, given an incomplete description and an instance of that concept which is erroneously not covered, requires to find a new description that covers this instance. Considering the *msc* of the instance as the description of the example at the language level, it is possible to compute their least common subsumer by transforming the concept descriptions into the corresponding clauses and then compute the *lgg* of the two clauses, as shown in [14]:

Theorem 5.1 (lcs using lgg) *Given two $\mathcal{ALN}$ descriptions C and D, it holds that* $lcs(C, D) = \Phi^{-1}(lgg(\Phi(C), \Phi(D)))$.

Other ILP algorithms that employ greedy bottom-up strategies for finding solutions to the learning problem may fall short because in DL each concept admits only one definition. Therefore, sometimes specialization steps are required by means of a suitable operator.

Now we consider the problem of computing specializations of a concept description D that erroneously covers a negative example N, w.r.t. the set of available positive examples in P. Again, it is assumed that the D is the clause obtained by transforming the related $\mathcal{ALN}$ concept description, and, similarly, the examples N and P_i stand here for the clauses obtained by transforming the (approximated) *msc*'s of the related assertions in the A-box. Since many incomparable specializations of a couple of descriptions are possible, the algorithm in Figure 1 describes how to compute the set of all the specializations. Then, the obtained clause should be transformed in an $\mathcal{ALN}$ concept description using Φ^{-1}. In the first phase, the *lgg* of the clauses corresponding to the positive examples is computed. This may turn out to be redundant and therefore requires its reduction. The obtained set of literals *CommonLits* represents the information that is common to all the positive examples and will be exploited as a source for the set of refinements *RefLits*. This set of literals is obtained in the main loop by comparing each literal selected from *CommonLits* at a given level to those in the negative clause N. When the selected literal L stands for a concept (or its negation) and it does not already occur in N, it is a candidate refinement thus it can be added to *RefLits*. On the other hand, when the selected literal L was produced by a role restriction, it can be added if no compatible literal occurs in N. Otherwise, the addition of a single literal is not sufficient to specialize D. If a compatible literal does not occur in D then the chosen literal L can be added but it cannot specialize the D ruling out the coverage of N. Yet, further linked literals can be added at the next level. Instead, when a compatible literal already occurs in D, one may consider the addition of this literal, possibly restricting the interval, provided that the interval is included in the one in the literal in D. Eventually, a specialization can be implemented by choosing the set of literals LL in *RefLits* that are connected to the clause D and then adding the set to the D (note that in case of a role restriction already occurring in D with a larger interval, the added literal must replace the other one.

Example 5.1 *Given the current description of a concept h, in the form of Horn clause:*

$$\begin{aligned}
C : h(X) \leftarrow\ & cp_{\mathsf{A}}(X), cp_{\mathsf{B}}(X),\\
&\quad rr_{\mathsf{S}}(X,[2,9],W), cp_{\mathsf{A}}(W), cp_{\mathsf{D}}(W),\\
&\qquad\quad rr_{\mathsf{S}}(W,[1,5],T), cp_{\mathsf{A}}(T),\\
&\quad rr_{\mathsf{T}}(X,[3,3],Z), cp_{\mathsf{C}}(Z), cp_{\mathsf{D}}(Z),\\
&\qquad\quad rr_{\mathsf{S}}(Z,[5,7],S), cp_{\mathsf{D}}(S).
\end{aligned}$$

($\mathsf{H} = \mathsf{A} \sqcap \mathsf{B} \sqcap \forall\mathsf{S}(\mathsf{A} \sqcap \mathsf{D} \sqcap \geq 2.\mathsf{S} \sqcap \leq 9.\mathsf{S} \sqcap \forall\mathsf{S}(\mathsf{A} \sqcap \geq 1.\mathsf{S} \sqcap \leq 5.\mathsf{S})) \sqcap \forall\mathsf{T}(\mathsf{C} \sqcap \mathsf{D} \sqcap \geq 3.\mathsf{T} \sqcap \leq 3.\mathsf{T} \sqcap \forall\mathsf{S}(\mathsf{D} \sqcap \geq 5.\mathsf{S} \sqcap \leq 7.\mathsf{S}))$*)*

and a negative instance it accounts for:

$$\begin{aligned}
E^{-} : \neg h(X) \leftarrow\ & cp_{\mathsf{A}}(X), cp_{\mathsf{B}}(X),\\
&\quad rr_{\mathsf{R}}(X,[4,4],Y), cp_{\mathsf{A}}(Y), cp_{\mathsf{B}}(Y), cp_{\mathsf{C}}(Y),\\
&\quad rr_{\mathsf{S}}(X,[5,5],W), cp_{\mathsf{A}}(W), cp_{\mathsf{C}}(W), cp_{\mathsf{D}}(W),\\
&\qquad\quad rr_{\mathsf{S}}(W,[2,2],T), cp_{\mathsf{A}}(T),\\
&\quad rr_{\mathsf{T}}(X,[3,3],Z), cp_{\mathsf{A}}(Z), cp_{\mathsf{C}}(Z), cp_{\mathsf{D}}(Z),\\
&\qquad\quad rr_{\mathsf{S}}(Z,[5,5],S), cp_{\mathsf{D}}(S).
\end{aligned}$$

($\neg\mathsf{H} = \mathsf{A} \sqcap \mathsf{B} \sqcap \forall\mathsf{R}(\mathsf{A} \sqcap \mathsf{B} \sqcap \mathsf{C} \sqcap \geq 4.\mathsf{R} \sqcap \leq 4.\mathsf{R}) \sqcap \forall\mathsf{S}(\mathsf{A} \sqcap \mathsf{C} \sqcap \mathsf{D} \sqcap \geq 5.\mathsf{S} \sqcap \leq 5.\mathsf{S} \sqcap \forall\mathsf{S}(\mathsf{A} \sqcap \geq 2.\mathsf{S} \sqcap \leq 2.\mathsf{S})) \sqcap \forall\mathsf{T}(\mathsf{A} \sqcap \mathsf{C} \sqcap \mathsf{D} \sqcap \geq 3.\mathsf{T} \sqcap \leq 3.\mathsf{T} \sqcap \forall\mathsf{S}(\mathsf{D} \sqcap \geq 5.\mathsf{S} \sqcap \leq 5.\mathsf{S}))$*)*

suppose the least general generalization of the past positive examples to be:

$$\begin{aligned}
G : h(X) \leftarrow\ & cp_{\mathsf{A}}(X), cp_{\mathsf{B}}(X), cp_{\mathsf{C}}(X),\\
&\quad rr_{\mathsf{R}}(X,[3,8],Y), cp_{\mathsf{A}}(Y), cp_{\mathsf{C}}(Y), cp_{\mathsf{D}}(Y),\\
&\quad rr_{\mathsf{S}}(X,[2,3],W), cp_{\mathsf{A}}(W), cp_{\mathsf{B}}(W), cp_{\mathsf{D}}(W),\\
&\qquad\quad rr_{\mathsf{S}}(W,[1,3],T), cp_{\mathsf{A}}(T),\\
&\quad rr_{\mathsf{T}}(X,[3,3],Z), cp_{\mathsf{C}}(Z), cp_{\mathsf{D}}(Z),\\
&\qquad\quad rr_{\mathsf{R}}(Z,[2,2],U), cp_{\mathsf{B}}(U),\\
&\qquad\qquad\quad rr_{\mathsf{T}}(U,[2,5],V), cp_{\mathsf{B}}(U),\\
&\qquad\quad rr_{\mathsf{S}}(Z,[5,5],S), cp_{\mathsf{A}}(S), cp_{\mathsf{D}}(S).
\end{aligned}$$

($\mathsf{H} = \mathsf{A} \sqcap \mathsf{B} \sqcap \mathsf{C} \sqcap \forall\mathsf{R}(\mathsf{A} \sqcap \mathsf{C} \sqcap \mathsf{D} \sqcap \geq 3.\mathsf{R} \sqcap \leq 8.\mathsf{R}) \sqcap \forall\mathsf{S}(\mathsf{A} \sqcap \mathsf{B} \sqcap \mathsf{D} \sqcap \geq 2.\mathsf{S} \sqcap \leq 3.\mathsf{S} \sqcap \forall\mathsf{S}(\mathsf{A} \sqcap \geq 1.\mathsf{S} \sqcap \leq 3.\mathsf{S})) \sqcap \forall\mathsf{T}(\mathsf{C} \sqcap \mathsf{D} \sqcap \geq 3.\mathsf{T} \sqcap \leq 3.\mathsf{T} \sqcap \forall\mathsf{R}(\mathsf{B} \sqcap \geq 2.\mathsf{R} \sqcap \leq 2.\mathsf{R} \sqcap \forall\mathsf{T}(\mathsf{B} \sqcap \geq 2.\mathsf{T} \sqcap \leq 5.\mathsf{T})) \sqcap \forall\mathsf{S}(\mathsf{A} \sqcap \mathsf{D} \sqcap \geq 5.\mathsf{S} \sqcap \leq 5.\mathsf{S}))$*)*

where the clause literals have been indented in order to reflect the concept nesting, and the same variable name is used in different clauses to remark the correspondence between literals across the clauses. Then, the list of all possible refinements is:

- *adding* $cp_{\mathsf{C}}(X)$*, found as an immediate restriction to the variable in the head (it belongs to all positive examples, appearing in their generalization, but not to the negative one);*
- *adding* $rr_{\mathsf{R}}(X,[3,8],Y), cp_{\mathsf{D}}(Y)$ *(the actual refinement is the second literal, since the former cannot exclude the negative example, and is included to ensure linkedness of the latter);*
- *replacing* $rr_{\mathsf{S}}(X,[2,9],W)$ *by* $rr_{\mathsf{S}}(X,[2,4],W)$ *(that shrinks the interval excluding the negative example and preserving the information in the* lgg*;conversely, shrinking the interval to* $rr_{\mathsf{S}}(X,[6,9],W)$ *would exclude also the* lgg*, and hence all the past positive examples represented by it);*
- *adding* $rr_{\mathsf{R}}(Z,[2,2],U)$ *(that is already linked to the current definition and is not present in the negative example, and hence is sufficient to avoid covering it);*
- *adding* $cp_{\mathsf{A}}(S)$*, since the interval in* $rr_{\mathsf{S}}(Z,[5,7],S)$ *cannot be shrunk to exclude the negative example while preserving the information in the least general generalization (indeed, their values are exactly the same); in this case, the role literal is inherited from the clause* C*, and not from the* lgg*, in order to keep the refinement minimal.*

Since any of such possibilities modifies the clause C *in a different variable, the resulting refinements are all incomparable to each other. Moreover, each of them is clearly minimal since only one literal is added or the interval has been shrunk just as needed to exclude the problematic value.*

```
specialization($D$,$N$,$P$): $D'$
input $D$: concept description;
      $N$: negative example;
      $P = \{P_1, \ldots, P_n\}$: positive examples
output$D'$: specialized concept description
begin
Let $\mu$ be such that $D\mu \subseteq N$;
(* Literals for producing refinements *)
$\mathrm{RefLits} \leftarrow \emptyset$;
$\mathrm{CommonLits} \leftarrow \mathrm{reduce}(lgg(P_1, \ldots, P_n))$;
(* those that are linked to the head *)
$\mathrm{Lits} \leftarrow \mathrm{linked}(\{h(X)\}, \mathrm{CommonLits})$; while $\mathrm{Lits} \neq \emptyset$ do
  begin
  for each $L \in \mathrm{Lits}$ do
    if ($\exists C$: $L = cp_C(X)$ or $L = cn_C(X)$) and $L \notin D$ and $L\mu \notin N$ then
      $\mathrm{RefLits} \leftarrow \mathrm{RefLits} \cup \{L\}$
    else (* $L = rr_R(X, [m_L, n_L], Y)$ *)
    if $L\mu \notin N$ then
      $\mathrm{RefLits} \leftarrow \mathrm{RefLits} \cup \{L\}$
    else (*more than one literal needed for specialization *)
      if $rr_R(X, [m_D, n_D], Y) \notin D$ then
        $\mathrm{RefLits} \leftarrow \mathrm{RefLits} \cup \{L\}$;
      else
        begin
        Let $L\mu = rr_R(X, [m_N, n_N], Y) \in N$
        Let $I'_1 = [m_L, m_N - 1]$ and $I'_2 = [n_N + 1, n_L]$
        if $I'_1 \not\subseteq [m_P, n_p]$ and $I'_2 \not\subseteq [m_P, n_p]$ then
          $\mathrm{RefLits} \leftarrow \mathrm{RefLits} \cup \{rr_R(X, [m_D, n_D], Y)\}$
        else
          $\mathrm{RefLits} \leftarrow \mathrm{RefLits} \cup \{rr_R(X, I'_k, Y)\}$
          with $k = 1, 2$ depending on the included interval
        end
  $\mathrm{Lits} \leftarrow \mathrm{linked}(\mathrm{Lits}, \mathrm{CommonLits})$
  end;
$LL \leftarrow \mathrm{select\_by\_linkedness}(\mathrm{RefLits}, D)$;
$D' \leftarrow \mathrm{implement}(LL, D)$
end
```

Figure 1. Specialization Algorithm.

The specialization algorithm is linear in the number of literals in the reduced *lgg*. Thus the most complex operation is really the reduction of this generalization. A simpler way to specialize an existing description can be obtained by considering only the literals in the incorrect clause D.

6. Conclusions and Future Work

Structural knowledge is expected to play an important role in the Semantic Web. Yet, while techniques for deductive reasoning for such knowledge bases are now very well assessed, their construction is a hard task for knowledge engineers even in limited domains; they could be supported by (semi-) automatic inductive tools. We have investigated the problem of learning concept definitions in DL both as search and presenting a method to induce and refine concept definitions. The method illustrated in this paper is currently being implemented using INTHELEX [11] as underlying ILP learning system that will induce or refine $\mathcal{ALN}$ knowledge bases. This will allow to design a learning and refinement service for the Semantic Web. The proposed framework could be extended along two directions. First, an investigation on the properties of the refinement operators on DL languages is required. In order to increase the efficiency of learning, redundancy during the search for solutions is to be avoided. This can be done by defining minimal refinement operators [3]. Secondly, the method can be extended to other DLs by changing the residual operator and devising a proper representation for counterfactuals.

7. Acknowledgements

This research was partially funded by the University of Bari, Project *"Tecniche di Apprendimento Automatico per l'Analisi di File di Log"*.

References

[1] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[2] F. Baader and R. Küsters. Computing the least common subsumer and the most specific concept in the presence of cyclic $\mathcal{ALN}$ concept descriptions. In O. Herzog and A. Güenter, editors, *Proceedings of the 22th Annual German Conference on Artificial Intelligence*, volume 1504 of *LNAI*, pages 129–140. Springer, 1998.

[3] L. Badea and S.-H. Nienhuys-Cheng. A refinement operator for description logics. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *LNAI*, pages 40–59. Springer, 2000.

[4] T. Berners-Lee. Design Issues: Technical and philosophical notes on web architecture, 1990-2002. http://www.w3.org/DesignIssues.

[5] A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1-2):353–367, 1996.

[6] W. Cohen and H. Hirsh. Learning the CLASSIC description logic. In P. Torasso, J. Doyle, and E. Sandewall, editors, *Proceedings of the 4th International Conference on the Principles of Knowledge Representation and Reasoning*, pages 121–133. Morgan Kaufmann, 1994.

[7] F. Donini, M. Lenzerini, D. Nardi, and W. Nutt. Tractable concept languages. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 458–463, Sydney, 1991.

[8] F. Donini, M. Lenzerini, D. Nardi, and M. Schaerf. $\mathcal{AL}$-log: Integrating Datalog and description logics. *Journal of Intelligent Information Systems*, 10:227–252, 1998.

[9] F. Esposito, N. Fanizzi, S. Ferilli, and G. Semeraro. A generalization model based on OI-implication for ideal theory refinement. *Fundamenta Informaticæ*, 47:15–33, 2001.

[10] F. Esposito, N. Fanizzi, S. Ferilli, and G. Semeraro. OI-implication: Soundness and refutation completeness. In B. Nebel, editor, *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 847–852, Seattle, WA, 2001.

[11] F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli. Multistrategy theory revision: Induction and abduction in INTHELEX. *Machine Learning*, 38(1/2):133–156, 2000.

[12] D. Haussler. Learning conjuntive concepts in structural domains. *Machine Learning*, 4:7–40, 1989.

[13] R. Khardon. Learning function-free Horn expressions. *Machine Learning*, 37(3):241–275, December 1999.

[14] J.-U. Kietz. Learnability of description logic programs. In S. Matwin and C. Sammut, editors, *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *LNAI*, pages 117–132, Sydney, 2002. Springer.

[15] B. Nebel. *Reasoning and Revision in Hybrid Representation Systems*, volume 422 of *LNAI*. Springer, 1990.

[16] S. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *LNAI*. Springer, 1997.

[17] S. Nienhuys-Cheng, W. Van Laer, J. Ramon, and L. De Raedt. Generalizing refinement operators to learn prenex conjunctive normal forms. In *Proceedings of the 9th International Conference on Inductive Logic Programming*, volume 1631 of *LNAI*, pages 245–256. Springer, 1999.

[18] C. Rouveirol and V. Ventos. Towards learning in CARIN-$\mathcal{ALN}$. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *LNAI*, pages 191–208. Springer, 2000.

[19] M. Sebag and C. Rouveirol. Constraint inductive logic programming. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 277–294. IOS Press, 1996.