Concept Formation in Expressive Description Logics

Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro

Dipartimento di Informatica, Università degli Studi di Bari
Via Orabona 4, 70125 Bari, Italy
surname@di.uniba.it

Abstract. We investigate on the automated construction of terminologies from assertions in an expressive Description Logics representation like $\mathcal{ALC}$. The overall unsupervised learning problem is decomposed into smaller supervised learning problems once clusters of disjoint concepts are detected. In turn, these problems are solved by means of refinement operators[1].

1 Motivation

In the perspective of the Semantic Web [1], an effort is required for supporting interoperability at a semantic level. Ontological knowledge is to be employed for organizing and classifying resources on the ground of their meaning. Such knowledge bases can be a powerful tool for supporting many other services, such as reasoning and retrieval. In the proposed frameworks, an ontology is cast as a concept graph, accounting for concepts and relationships, in specific or larger contexts, intended for being used by machines. Each class of resources is defined extensionally by the set of the resources it represents, and intensionally by descriptions which account for them and possibly also for instances that may be available in the future. Annotating resources after semantic criteria is not a trivial and inexpensive task. Thus, the problem is how to support the construction of such ontological knowledge. In this context, we focus on the induction of definitions for classes of resources from their available assertions. Indeed, supervised maintenance tools can be an important factor to boost the realization of the Semantic Web. In a learning service for the Semantic Web, representation languages that are typical in this context have to be considered. Such languages are closely related to the family of languages known as *Description Logics* (henceforth DL), which are endowed with well founded semantics and reasoning procedures [2]. In DL knowledge bases, the *world state* (extension) is given by an *A-box* to be regarded as a collection of assertions about the resources, while the structural descriptions of their classes (intension) are maintained in a *T-box*. The induction of structural knowledge, like the T-box taxonomies, is not new in machine

[1] This research was partially funded by the European Commission under the IST Integrated Project VIKEF - Virtual Information and Knowledge Environment Framework (Contract no. 507173); more information at http://www.vikef.net.

J.-F. Boulicaut et al. (Eds.): ECML 2004, LNAI 3201, pp. 99–110, 2004.

learning, especially in the context of *concept formation* where clusters of similar objects are aggregated in hierarchies according to heuristic criteria or similarity measures. Most of the methods apply to simple (propositional or equivalent) representations, whereas ontologies require richer structural languages. Yet, the induction of structural knowledge turns out to be a hard problem in first-order logic representations or fragments therein [3]. In *Inductive Logic Programming* (ILP), attempts have been made to extend relational learning techniques towards more expressive languages [4] or hybrid representations [5]. In the DL literature, often inductive methods are based on a heuristic search to cope with the problem complexity. They generally implement bottom-up operators, such as the *least common subsumer* defined for various DL languages [6], that tend to induce correct yet overly specific concept definitions that may have poor predictive capabilities. Moreover, for the sake of efficiency, simple DLs have been taken into account which are less expressive than the current standards for ontology markup languages. In the proposed methodology, an expressive language like $\mathcal{ALC}$ [2] is adopted. As in previous work [7], though on different representations, the overall unsupervised learning problem is decomposed into smaller supervised learning problems. Initially, a basic taxonomic structure of the search space is induced from the A-box. This phase elicits clusters of mutually disjoint concepts that require a discriminant definition. Thus, several smaller supervised learning problems are issued. This task is cast as a search for correct definitions for a concept in the context of its cluster(s). Therefore, investigating the properties of the search space and the related operators, we also define a method for induction and refinement in $\mathcal{ALC}$. This method, relying on the notion of counterfactuals [8], can exploit more intensively the knowledge provided by the available assertions. The paper is organized as follows. In Sec. 2 the search space and its properties are presented. The method for knowledge base induction is illustrated and discussed in Sect. 3. Sect. 4 concludes examining possible extensions.

2 Preliminaries on the Search Space

The theoretical setting of learning in DL spaces requires the definition of syntax and semantics for the proposed representation. The data model should be expressed by means of DL concept languages for which reasoning tools are already available. Furthermore, the learning problem is cast as a search in the space of candidate definitions induced by the reference representation.

2.1 Knowledge Bases in Description Logics

In a DL language [2], primitive *concepts* $N_C = \{C, D, \ldots\}$ are interpreted as subsets of a certain domain of objects and primitive *roles* $N_R = \{R, S, \ldots\}$ are interpreted as binary relations. In $\mathcal{ALC}$ [2] complex descriptions can be built from primitive concepts and roles by means of the constructors given in Tab. 1. In an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, $\Delta^{\mathcal{I}}$ is the domain of the interpretation and the functor $\cdot^{\mathcal{I}}$ maps the concepts to their extensions (subsets of the domain $\Delta^{\mathcal{I}}$).

Table 1. $\mathcal{ALC}$ constructors and related interpretation.

Constructor	Syntax	Semantics $\mathcal{I}$
top concept	$\top$	$\Delta^{\mathcal{I}}$
bottom concept	$\bot$	$\emptyset$
concept negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
concept conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
concept disjunction	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
value restriction	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y\ (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
existential restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y\ (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$

A *knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ contains a T-box $\mathcal{T}$ and an A-box $\mathcal{A}$. $\mathcal{T}$ is a set of (acyclic) concept definitions $C \doteq D$, meaning $C^{\mathcal{I}} = D^{\mathcal{I}}$, where C is the concept name and D is a DL description given in terms of the language constructors. $\mathcal{A}$ 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 $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. A notion of *subsumption* between concepts (or roles) is given in terms of their interpretations:

Definition 2.1 (subsumption). *Given two concept (role) definitions D_1 and D_2 in a T-box $\mathcal{T}$, D_1* subsumes *D_2 iff $D_1^{\mathcal{I}} \supseteq D_2^{\mathcal{I}}$ holds for every interpretation $\mathcal{I}$ of $\mathcal{T}$. This is denoted by $D_1 \sqsupseteq_{\mathcal{T}} D_2$, or simply $D_1 \sqsupseteq D_2$ when $\mathcal{T}$ is obvious.*

Example 2.1. An example of concept definition in the proposed language:
$\texttt{Father} \doteq \texttt{Male} \sqcap \forall \texttt{hasChild.Being} \sqcap \exists \texttt{hasChild.Being}$
which translates the sentence *"fathers are male that have beings as their children, and precisely at least one child"*. A specialized concept definition is:
$\texttt{FatherWithoutSons} \doteq \texttt{Male} \sqcap \exists \texttt{hasChild.Being} \sqcap \forall \texttt{hasChild.}(\texttt{Being} \sqcap \neg \texttt{Male})$
It holds that: $\texttt{Father} \sqsupseteq \texttt{FatherWithoutSons}$.
Examples of A-box assertions are the following:
$\texttt{Father}(\texttt{zeus}), \texttt{Male}(\texttt{zeus}), \texttt{God}(\texttt{zeus}), \neg\texttt{Father}(\texttt{era}), \texttt{hasChild}(\texttt{zeus}, \texttt{apollo}),$
$\texttt{hasChild}(\texttt{zeus}, \texttt{hercules}), \neg\forall \texttt{hasChild.God}(\texttt{zeus})$

A concept may have many semantically equivalent, yet syntactically different, descriptions that can be reduced to a normal form by means of equivalence-preserving rewriting rules (see [2]). Preliminarily, the different parts of a description are to be designated: $\mathsf{prim}(C)$ is the set of the concepts at the top-level conjunction of C; if there exists a universal restriction $\forall R.C'$ at the top-level of C then $\mathsf{val}_R(C) = C'$ otherwise $\mathsf{val}_R(C) = \top$; $\mathsf{ex}_R(C)$ is the set of the descriptions C' in existential restrictions $\exists R.C'$ at the top-level conjunction of C.

Definition 2.2 (normal form). *An $\mathcal{ALC}$ concept description D is in* normal form *iff $D \equiv \bot$ or $D \equiv \top$ or if $D = D_1 \sqcup \cdots \sqcup D_n$ $(\forall i \in \{1, \ldots, n\}\ D_i \not\equiv \bot)$ with*

$$D_i = \bigsqcap_{A \in \mathsf{prim}(D_i)} A \quad \sqcap \bigsqcap_{R \in N_R} \left[\bigsqcap_{V \in \mathsf{val}_R(D_i)} \forall R.V \quad \sqcap \bigsqcap_{E \in \mathsf{ex}_R(D_i)} \exists R.E \right]$$

where and $\forall R \in N_R$ every description in $\mathsf{ex}_R(D_i) \cup \mathsf{val}_R(D_i)$ is in normal form.

2.2 Induction as Search

Provided that an order is imposed on the concept descriptions, the induction of the definitions for undefined concepts in the A-box can be cast as a search process. The *search space* depends on the order adopted that induces a generalization model. This provides a criterion for traversing the space of solutions by means of suitable operators [9]. They allow for the treatment of induction as a search process that is decoupled from the specific heuristics to be employed.

Definition 2.3 (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 operator* τ *for* $C \in \mathcal{S}$ *is defined:* $\tau^*(C) = \bigcup_{n \geq 0} \tau^n(C)$*, where* $\tau^0(C) = \{C\}$ *and* $\tau^n(C) = \{D \in \mathcal{S} \mid \exists E \in \tau^{n-1}(C) \colon\ D \in \tau(E)\}$.

The properties of the operators depend on the order adopted.

Definition 2.4 (properties). *In a quasi-ordered set* $(\mathcal{S}, \succeq)$*, a refinement operator* τ *is* locally finite *iff* $\forall C \in \mathcal{S} : \tau(C)$ *is finite and computable. A downward (resp. upward) refinement operator* ρ *(resp.* δ*) is* proper *iff* $\forall C \in \mathcal{S} \colon\ D \in \rho(C)$ *implies* $C \succ D$ *(resp.* $D \in \delta(C)$ *implies* $D \succ C$*). A downward (resp. upward) refinement operator* ρ *(resp.* δ*) is* complete *iff* $\forall C, D \in \mathcal{S}$*,* $D \succ C$ *implies* $\exists E \in \rho^*(D)$ *such that* $E \equiv C$ *(resp.* $C \succ D$ *implies* $\exists E \in \delta^*(D)$ *such* $E \equiv C$*). A locally finite, proper and complete operator is defined as* ideal.

In inductive reasoning, it is necessary to test the coverage of candidate hypotheses with respect to the examples. Coverage also determines the decisions on the possible refinement of such hypotheses. However, it should be noted that in the DL settings the *Open World Assumption* (OWA) is adopted, differently from the context of learning or query answering where the *Closed World Assumption* (CWA) is commonly made. It is supposed that preliminarily a representative at the concept language level is derived in the form of *most specific concept* (*msc*) [6, 2]. The *msc* required by the following algorithms is a concept description, entailing the given assertion, that it is bound to be among the most specific ones. Hence, the examples will be represented with very specific conjunctive descriptions obtained by means of the *realization* [2] of the assertions. Since an *msc* need not to exist in $\mathcal{ALC}$, for each individual, we consider an approximation of its *msc* with respect to $\mathcal{A}$, up to a certain depth k [2]:

Definition 2.5 (coverage). *Given the knowledge base* $\langle \mathcal{T}, \mathcal{A} \rangle$*, the definition of a concept* $C \doteq D$ covers *an assertion* $C(e)$ *iff* $\exists k \colon\ D \sqsupseteq_{\mathcal{T}} msc_{k,\mathcal{A}}(e)$

The unsupervised learning problem can be formally stated as follows:

Definition 2.6 (DL learning problem). *In a search space* $(\mathcal{S}, \sqsupseteq)$
Given *a knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$*, supposed that* $\mathcal{T}$ *does not contain definitions for all the concepts with assertions in* $\mathcal{A}$
Induce *a set of concept definitions, by means of refinement operators,* $\mathcal{T}_C = \{C_1 \doteq D_1,\ C_2 \doteq D_2,\ \ldots\}$ *such that* $\forall i \forall C_i(e) \in \mathcal{A} \colon\ \mathcal{T} \cup \mathcal{T}_C$ *covers* $C_i(e)$

The problem requires to find definitions $\mathcal{T}_C$ for undefined concepts concerning assertions in the A-box. $\mathcal{T}$ can be regarded as a sort of background knowledge (possibly imported from higher level ontologies), which is supposed to be correct but also incapable to account for all of the assertions in the A-box.

3 Induction of $\mathcal{ALC}$ Concept Descriptions

A method for concept formation in the referred representation is presented[2]. After inferring the basic structure of the search space, the concept characterization requires the solution of supervised learning problems. Albeit these can be solved by means of refinement operators and heuristics, a more knowledge-intensive method based on counterfactuals is used to increase the process efficiency.

3.1 A Concept Formation Algorithm

The main concept formation algorithm applied in our method is presented in Fig. 1. As mentioned, it consists of two phases: construction of the basic taxonomy and solution of the induced learning problems.

The basic taxonomy of primitive concepts is built from the knowledge available in the starting A-box $\mathcal{A}$: if also $\mathcal{T}$ is non-empty, then $\mathcal{A}$ is augmented with assertions obtained by saturation with respect to $\mathcal{T}$. This phase also singles out domains and ranges of the roles as the starting superconcepts SCs (step 5). Direct subsumption relationships between the concept extensions are detected (*level-wise*) so to build up a hierarchy based on them. Meanwhile, the relationships of pairwise disjointness among the subconcepts Cs_j that can induced from the A-box are exploited to infer the candidate clusters of concepts MDCs (steps 6–12). A MDC stands for the maximal set of *mutually disjoint concepts*, i.e. a cluster of disjoint subconcepts of the same concept. MDCs are built by iteratively splitting by disjointness the sets of (direct) subconcepts of a superconcept SC_j. At each iteration, the set of the subconcepts replaces the current set of superconcepts SCs for the next level (until a base level is reached). However, finding the MDCs has a superpolynomial worst case complexity [7].

Mutually disjoint concepts within an MDC require non-overlapping definitions. This is aimed at during the supervised phase (steps 13–25). Each non-primitive subconcept C_i in the selected MDC needs a discriminating definition that is induced as the result of a separate supervised learning problem, where the instances of disjoint subconcepts in the MDC act as negative examples. Thus, a loop is repeated looking for a candidate definition D_i for each concept in the selected MDC using an upward operator δ. A disjoint per loop is calculated covering (part of) the positive instances $\mathcal{A}_i$ related to concept C_i. When a definition D_i covers negative examples represented by the instances of other concepts in the context of the MDC, it has to be specialized by a downward operator. A given threshold $\min_q$ states the minimum quality for a candidate definition D_i.

[2] It was inspired by KLUSTER [7] where the BACK language [2] is adopted.

Tbox-induction($\mathcal{A}$,$\mathcal{T}$,$\mathcal{T}_C$)
input $\mathcal{A}$: A-box; $\mathcal{T}$: T-box
output $\mathcal{T}_C$: T-box
1. **if** $\mathcal{T} \neq \emptyset$ **then** $\mathcal{A} \leftarrow \mathcal{A} \cup \text{saturate}(\mathcal{A},\mathcal{T})$
2. **for each** primitive concept C_i with assertions in $\mathcal{A}$ **do**
3. $D_i \leftarrow \bot$
4. $\mathcal{A}_i \leftarrow \{C_i(e) \in \mathcal{A}\}$
5. $SCs \leftarrow \{domain(R) \mid R \in N_R(\mathcal{A})\} \cup \{range(R) \mid R \in N_R(\mathcal{A})\}$
6. **repeat** (* find the MDCs *)
7. **for each** $SC_j \in SCs$ **do**
8. $Cs_j \leftarrow \{C \sqsubset SC_j \mid \nexists C' : C \sqsubset C' \sqsubset SC_j\}$
9. $MDCs_j \leftarrow \text{split\_disjoint}(Cs_j)$
10. $MDCs \leftarrow MDCs \cup MDCs_j$
11. $SCs \leftarrow \bigcup_j Cs_j$
12. **until** $SCs = \emptyset$ (* no direct subconcepts *)
13. **repeat** (* supervised phase *)
14. $MDC \leftarrow \text{select}(MDCs)$
15. **repeat** (* solve next supervised learning problem *)
16. choose $C_i \in MDC$ such that $\mathcal{A}_i \neq \emptyset$
17. $D_i \leftarrow D_i \sqcup \text{generalize}(D_i,\mathcal{A}_i,\delta)$
18. $q \leftarrow \text{eval}(D_i, MDC)$
19. **while** $q < min_q$ **and** refinable(C_i) **do**
20. $D_i \leftarrow \text{specialize}(D_i, MDC, \rho)$
21. $q \leftarrow \text{eval}(D_i, MDC)$
22. $\mathcal{A}_i \leftarrow \mathcal{A}_i \setminus \{C_i(e) \in \mathcal{A}_i \mid D_i \text{ covers } C_i(e)\}$
23. **until** $\forall C_i \in MDC$: $\mathcal{A}_i = \emptyset$
24. $MDCs \leftarrow MDCs \setminus MDC$
25. **until** $MDCs = \emptyset$
26. $\mathcal{T}_C \leftarrow \{C_i \doteq D_i \mid \textit{for each concept}\}$
27. **return** $\mathcal{T}_C$

Fig. 1. The main T-box induction algorithm.

Like in the *lcs*-based approach [6], the initial characterization of a concept is modeled like a bottom-up search for generalizations. In [9] a different strategy is proposed. The search should start from the most general definition $\top$, and then it would repeatedly apply a downward refinement operator ρ up to finding discriminating generalizations for the target concepts. The method described in [7] employs incomplete specialization and generalization operators. It is not guaranteed to find a correct definition, since the search space is limited in order to preserve efficiency. For example, the generalization algorithm follows a predefined schema that forces an order in the refinement graph which may not lead to the correct definitions.

3.2 Refinement Operators for $\mathcal{ALC}$ and Heuristics

Given the ordering relationship induced by subsumption for the space of hypotheses $(\mathcal{ALC}, \sqsupseteq)$, it is possible to specify how to traverse this space by means

of refinement operators. Preliminarily, the definition of a difference operator is needed for both conjunctive and disjunctive descriptions. In the former case, $C = C_1 \sqcap \cdots \sqcap C_n$, the difference is the generalized conjunct resulting from removing one conjunct: $C - C_i = \bigsqcap_{k \neq i} C_k$. In latter case, $D = D_1 \sqcup \cdots \sqcup D_m$, the difference is the specialized disjunct $D - D_j = \bigsqcup_{k \neq j} D_k$. Considered the $\mathcal{ALC}$ normal form, each level of a concept description interleaves disjunctive or conjunctive descriptions. Thus, the operators should accommodate either case.

Definition 3.1 (downward operator). *In the search space* $(\mathcal{ALC}, \sqsupseteq)$*, the* downward refinement operator $\rho_\sqcup$ *for disjunctive concept descriptions (in* $\mathcal{ALC}$ *normal form)* $D = D_1 \sqcup \cdots \sqcup D_n$ *is defined as follows:*

- $D' \in \rho_\sqcup(D)$ *if* $D' = D - D_i$ *for some* $1 \leq i \leq n$
- $D' \in \rho_\sqcup(D)$ *if* $D' = (D - D_i) \sqcup D'_i$ *for some* $D'_i \in \rho_\sqcap(D_i),\ 1 \leq i \leq n$

The downward refinement operator $\rho_\sqcap$*, given a conjunctive concept description* $C = C_1 \sqcap \cdots \sqcap C_m$*, is defined as follows:*

- $C' \in \rho_\sqcap(C)$ *if* $C' = C \sqcap C_{j+1}$ *for some* $C_{j+1} \not\sqsupseteq C$
- $C' \in \rho_\sqcap(C)$ *if* $C' = (C - C_j) \sqcap C'_j$ *for some* $1 \leq j \leq m$*, where:*
  - $C'_j = \exists R.D'_j$, $C_j = \exists R.D_j$ *and* $D'_j \in \rho_\sqcup(D_j)$ *or*
  - $C'_j = \forall R.D'_j$, $C_j = \forall R.D_j$ *and* $D'_j \in \rho_\sqcup(D_j)$

It is straightforward to define the dual upward operator that seeks for more general hypotheses by adding disjuncts or refining them.

Definition 3.2 (upward operator). *In the search space* $(\mathcal{ALC}, \sqsupseteq)$*, the* upward refinement operator $\delta_\sqcup$ *for disjunctive concept descriptions (in* $\mathcal{ALC}$ *normal form)* $D = D_1 \sqcup \cdots \sqcup D_n$ *is defined as follows:*

- $D' \in \delta_\sqcup(D)$ *if* $D' = D \sqcup D_{n+1}$ *for some* D_{n+1} *such that* $D_{n+1} \not\sqsubseteq D$
- $D' \in \delta_\sqcup(D)$ *if* $D' = (D - D_i) \sqcup D'_i$ *for some* $D'_i \in \delta_\sqcap(D_i),\ 1 \leq i \leq n$

The upward refinement operator $\delta_\sqcap$*, given a conjunctive concept description* $C = C_1 \sqcap \cdots \sqcap C_m$*, is defined:*

- $C' \in \delta_\sqcap(C)$ *if* $C' = C - C_j$ *for some* $1 \leq j \leq m$
- $C' \in \delta_\sqcap(C)$ *if* $C' = (C - C_j) \sqcap C'_j$ *for some* $1 \leq j \leq m$*, where:*
  - $C'_j = \exists R.D'_j$, $C_j = \exists R.D_j$ *and* $D'_j \in \delta_\sqcup(D_j)$ *or*
  - $C'_j = \forall R.D'_j$, $C_j = \forall R.D_j$ *and* $D'_j \in \delta_\sqcup(D_j)$

It can be shown that these operators are complete although highly redundant and therefore non ideal. Ideal refinement operators have been proven not to exists in spaces where infinite chains of descriptions occur [10]. In our case, one can consider the infinite chain $\exists R.\top \sqsupseteq \exists R.\exists R.\top \sqsupseteq \exists R.\exists R.\exists R.\top$, etc... Owing to the large extent of the search space, heuristics should be used together with refinement operators in order to focus their search. Defining suitable heuristics based on the available assertions can address a refinement operator to promising regions of the search space. The candidate hypotheses evaluation should take into

account the coverage of positive and negative examples in the learning problem. Intuitively, a good hypothesis should cover as many positive examples as possible and reject the negative ones. Moreover, other limitations are typically made upon the size of the hypotheses, in favor of the simpler ones (those containing less restrictions and with the lowest nesting factor, in the adopted language). In the algorithm presented, a possible form for the evaluation function of the i-th concept definition with respect to the j-th MDC may be:

$$\text{eval}(D_i, MDC_j) = w_p \cdot pos_{ij} - w_n \cdot neg_{ij} - w_s \cdot size_i$$

here pos_{ij} and neg_{ij} are determined by the rate of examples covered by the candidate hypothesis over the examples in the MDC, while $size_i$ should be calculated on the ground of its syntactic complexity (each term adds 1 to the size plus the size of the nested concept description, if any). Although this approach may seem quite simplistic, it has proven effective in an ILP context. It should be noted that subsumption in $\mathcal{ALC}$ is computationally expensive and it is known to be hardly reducible to structural operations [2]. Therefore, pursuing a mere *generate and test* strategy for solving supervised learning problems, even investigating better operators, is bound to be inefficient. Better methods should be devised for solving supervised learning problems. The information conveyed by the assertions is to be exploited in the refinement process rather than being tested afterwards.

3.3 Solving Supervised Learning Problems

A learning methodology is proposed based on the notion of counterfactuals built on the ground of residual learning problems [8]. The learning algorithm relies on two interleaving routines performing, respectively, generalization and specialization, that call each other for converging toward a correct concept definition.

The generalization algorithm (Fig. 2) is a greedy covering one: it tries to define positive examples by constructing disjunctive definitions. At each outer iteration, the msc of an example is selected as a starting seed for a new partial generalization; then, iteratively, the hypothesis is generalized by means of the operator δ (with a heuristic that privileges the refinements that cover the most of positives) until all positive concept representatives are covered or some negative representatives are explained. In such a case, the current concept definition *ParGen* has to be specialized by some counterfactuals. The co-routine, which receives the covered examples as its input, finds a sub-description K that is capable of ruling out the negative examples previously covered.

In the specializing routine (Fig. 3), given a previously computed hypothesis *ParGen*, which is supposed to be complete yet inconsistent with respect to some negative assertions, it must find counterfactuals that, conjuncted to the incorrect definition, can restore its correctness by ruling out the covered negative instances. The algorithm is based on the construction of residual learning problems based on the sub-descriptions that caused the subsumption of the negative examples, represented by their msc's. In this case, a residual is derived by considering that part of the incorrect definition *ParGen* that did not play a role in the

generalize(*Positives*, *Negatives*, *Generalization*)
input *Positives*, *Negatives*: positive and negative instances at concept level;
output *Generalization*: generalized concept definition
1. $ResPositives \leftarrow Positives$
2. $Generalization \leftarrow \bot$
3. **while** $ResPositives \neq \emptyset$ **do**
4. $ParGen \leftarrow$ select_seed($ResPositives$)
5. $CoveredPos \leftarrow \{Pos \in ResPositives \mid ParGen \sqsupseteq Pos\}$
6. $CoveredNeg \leftarrow \{Neg \in Negatives \mid ParGen \sqsupseteq Neg\}$
7. **while** $CoveredPos \neq ResPositives$ **and** $CoveredNeg = \emptyset$ **do**
8. $ParGen \leftarrow \text{select}(\delta(ParGen), ResPositives)$
9. $CoveredPos \leftarrow \{Pos \in ResPositives \mid ParGen \sqsupseteq Pos\}$
10. $CoveredNeg \leftarrow \{Neg \in Negatives \mid ParGen \sqsupseteq Neg\}$
11. **if** $CoveredNeg \neq \emptyset$ **then**
12. $K \leftarrow$ **specialize**($ParGen$, $CoveredPos$, $CoveredNeg$)
13. $ParGen \leftarrow ParGen \sqcap \neg K$
14. $Generalization \leftarrow Generalization \sqcup ParGen$
15. $ResPositives \leftarrow ResPositives \setminus CoveredPos$
16. **return** *Generalization*

Fig. 2. The generalizing routine.

subsumption. The residual will be successively employed as a positive instance of that part of description that should be ruled out of the definition (through negation). Analogously, the msc's derived from positive assertions plays the opposite role of negative instances for the residual learning problem under construction. Finally, this problem is solved by conjoining the negation of the generalization returned by the co-routine, applying it to these example descriptions.

The refinement method is somehow *specialization-oriented* being the generalization mechanism weaker than the one used for specializing the concept definitions. Thus, for the algorithm to work it is required that the starting approximations calculated on the ground of the negative examples be not greater than those relative to the positive ones. The function for calculating residuals is essentially a difference function. In the case of the $\mathcal{ALC}$ language, it is easy to define it as $C - D \equiv C \sqcup \neg D$.

Example 3.1. Suppose that the starting A-box is[3] $\mathcal{A} = \{M(d), r(d,l), r(j,s), \neg M(m), r(m,l), \neg M(a), w(a,j), r(a,s), F(d), F(j), \neg F(m) \neg F(a)\}$.

Let F be the target concept, thus the examples and counterexamples are: $Positives = \{d, j\}$ and $Negatives = \{m, a\}$. The approximated msc's are:

$msc(j) = \exists r.\top$ $\quad$ $msc(d) = M \sqcap \exists r.\top$
$msc(m) = \neg M \sqcap \exists r.\top$ $\quad$ $msc(a) = \neg M \sqcap \exists r.\top \sqcap \exists w.\top$

In Fig. 4 learning process step-wise run is reported. The result is $F = M \sqcap \exists r.\top$.

It can be proven that, provided that the adopted language bias is appropriate, the method actually converges to a solution of the learning problem [11]:

[3] F stands for `Father`, M for `Man`, r for the `parentOf` role while w represents `wifeOf`.

specialize($ParGen$, $CoveredPos$, $CoveredNeg$, K)
input $ParGen$: inconsistent concept definition
$CoveredPos$, $CoveredNeg$: covered positive and negative descriptions
output K: counterfactual
1. $NewPositives \leftarrow \emptyset$
2. $NewNegatives \leftarrow \emptyset$
3. **for each** $N_i \in CoveredNeg$ **do**
4. $NewP_i \leftarrow \text{residual}(N_i, ParGen)$
5. $NewPositives \leftarrow NewPositives \cup \{NewP_i\}$
6. **for each** $P_j \in CoveredPos$ **do**
7. $NewN_j \leftarrow \text{residual}(P_j, ParGen)$
8. $NewNegatives \leftarrow NewNegatives \cup \{NewN_j\}$
9. $K \leftarrow$ **generalize**($NewPositives$, $NewNegatives$)
10. **return** K

Fig. 3. The specializing routine.

Theorem 3.1 (correctness). *The algorithm eventually terminates computing a correct concept definition when this exists.*

The process terminates because the *generalize* routine produces one disjunct at each outer iteration, accommodating at least one positive example which is successively removed. Then the termination of the routine is guaranteed, provided that the inner loop terminates. This loop generalizes the seed-definition by applying δ. Eventually, it terminates either because the generalization is correct or it covers some negative example. This is the case when the *specialize* co-routine is invoked. This routine contains two initial loops that are controlled by the sizes of the input example sets. Each loop produces a residual concept definition to be used for the successive generalization. Provided that the other routine produces a correct generalization of the residual concept then also the specializing routine terminates. As regards the correctness, it is to be proven that, on return from the routine, the following relations hold: 1. $\forall Pos \in CoveredPos$: $(ParGen \sqcap \neg K) \sqsupseteq Pos$ and 2. $\forall Neg \in Negatives$: $(ParGen \sqcap \neg K) \not\sqsupseteq Neg$. If the call to *generalize* succeeds then for all $NewN_j \in NewNegatives$: $K \not\sqsupseteq NewN_j$. By definition, $P_j \in Positives$ implies that residual$(P_j, ParGen) \in NewNegatives$. Hence $K \not\sqsupseteq$ residual$(P_j, ParGen)$. Since $\forall Pos \in Positives = CoveredPos$: $ParGen \sqsupseteq Pos$ condition 1. holds. For condition 2., recall that $(ParGen \sqcap \neg K) \not\sqsupseteq Neg$ iff $ParGen \not\sqsupseteq Neg$ or $ParGen \sqsupseteq Neg$ and $K \sqsupseteq$ residual$(Neg, ParGen)$. Considering all $Neg \in Negatives$, if $ParGen \not\sqsupseteq Neg$ then the condition holds. Otherwise, if $ParGen \sqsupseteq Neg$ then the routine builds a residual element residual$(Neg, ParGen)$ for $NewPositives$. Thus, on return from *generalize*, K is a generalization of every description in $NewPositives$, and hence $K \sqsupseteq$ residual$(Neg, ParGen)$. Here, it is assumed that the language bias is adequate for the target problem otherwise the algorithm would fail to discriminate between identical descriptions. In this case, new concepts or roles may be introduced (*constructive induction*) for building discriminating definitions [7]. This goes beyond the scope of our present work. The specialization routine is linear in the number of examples except for the

generalize:
ResidualPositives ← {msc(d), msc(j)}
Generalization ← ⊥
/* Outer while loop */
ParGen ← $msc(d) = M \sqcap \exists r.\top$
CoveredPos ← $\{msc(d)\}$ and CoveredNeg ← {}
ParGen ← $\exists r.\top$ /* M dropped in the inner loop step 8.*/
CoveredPos ← $\{msc(d), msc(j)\}$ and CoveredNeg ← $\{msc(m), msc(a)\}$
Call **specialize**$(\exists r.\top, \{msc(d), msc(j)\}, \{msc(m), msc(a)\})$
specialize:
NewP$_1$ ← $\neg M \sqcap \exists r.\top \sqcup \neg\exists r.\top = \neg M$
NewPositives ← $\{\neg M\}$
NewP$_2$ ← $\neg M \sqcap \exists r.\top \sqcap \exists w.\top \sqcup \neg(\exists r.\top) = \neg M \sqcap \exists w.\top$
NewPositives ← $\{\neg M, \neg M \sqcap \exists w.\top\}$
NewN$_1$ ← $M \sqcap \exists r.\top \sqcup \neg\exists r.\top = M$
NewNegatives ← $\{M\}$
NewN$_2$ ← $\top$
NewNegatives ← $\{M, \top\}$
Call **generalize**$(\{\neg M, \neg M \sqcap \exists w.\top\}, \{M, \top\})$
generalize:
ResidualPositives ← $\{\neg M, \neg M \sqcap \exists w.\top\}$
Generalization ← ⊥
/* Outer while loop */
ParGen ← $\neg M \sqcap \exists w.\top$
CoveredPos ← $\{\neg M \sqcap \exists w.\top\}$ and CoveredNeg ← {}
/* Second while loop*/
ParGen ← $\neg(M \sqcup \neg(M \sqcap \exists w.\top) = \neg M$
CoveredPos ← $\{\neg M, \neg M \sqcap \exists w.\top\}$ and CoveredNeg ← {}
Generalization ← $\neg M$
Return $\neg M$
Return $\neg M$ /* back to first call to Generalization */
ParGen ← $\exists r.\top \sqcap \neg(\neg M)$
Generalization ← $\exists r.\top \sqcap \neg(\neg M)$
ResidualPositives ← {}
Return $\exists r.\top \sqcap M$
$F = M \sqcap \exists r.\top$

Fig. 4. The learning process for the A-Box in Ex. 3.1.

dependency on the generalization algorithm. Then it suffices here to discuss the complexity of such an algorithm. The generalization proposed here is a generic divide and conquer algorithm which performs a greedy search using the refinement operator δ. The number of iterations is linear in the number of instances. The actual source of complexity are the subsumption tests which are known to be PSpace-complete in $\mathcal{ALC}$ [2].

4 Conclusions and Future Work

While deductive reasoning and querying for knowledge bases in DL representation are well assessed, their construction can be a complex task for knowledge engineers which calls for semi-automatic tools. A method for concept formation in the $\mathcal{ALC}$ description logic has been presented. We applied a two-step algorithm where the first step induces level-wise supervised learning problems solved in the second step. The learning process can be performed by using the proposed refinement operators and heuristics. Besides, we have developed a more efficient method based on counterfactuals learning from the available assertions. The proposed framework could be extended along three directions. First, a more

expressive language bias could be chosen: e.g. the transitivity of relations would allow to learn recursive concepts. Secondly, inductive construction should be employed to introduce new concept and roles that help the definition and organization of the knowledge bases. Finally, we want to investigate learning with hybrid representations, where clausal logic descriptions are mixed with description logics, the latter accounting for the available ontological knowledge.

References

1. Berners-Lee, T.: Design Issues: Technical and philosophical notes on web architecture (1990-2002) http://www.w3.org/DesignIssues.
2. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P., eds.: The Description Logic Handbook. Cambridge University Press (2003)
3. Haussler, D.: Learning conjuntive concepts in structural domains. Machine Learning **4** (1989) 7–40
4. Nienhuys-Cheng, S., Laer, W.V., Ramon, J., Raedt, L.D.: Generalizing refinement operators to learn prenex conjunctive normal forms. In: Proceedings of the International Conference on Inductive Logic Programming. Volume 1631 of LNAI., Springer (1999) 245–256
5. Rouveirol, C., Ventos, V.: Towards learning in CARIN-$\mathcal{ALN}$. In Cussens, J., Frisch, A., eds.: Proceedings of the 10th International Conference on Inductive Logic Programming. Volume 1866 of LNAI., Springer (2000) 191–208
6. Cohen, W., Hirsh, H.: Learning the CLASSIC description logic. In Torasso, P., Doyle, J., Sandewall, E., eds.: Proceedings of the 4th International Conference on the Principles of Knowledge Representation and Reasoning, Morgan Kaufmann (1994) 121–133
7. Kietz, J.U., Morik, K.: A polynomial approach to the constructive induction of structural knowledge. Machine Learning **14** (1994) 193–218
8. Vere, S.: Multilevel counterfactuals for generalizations of relational concepts and productions. Artificial Intelligence **14** (1980) 139–164
9. Badea, L., Nienhuys-Cheng, S.H.: A refinement operator for description logics. In Cussens, J., Frisch, A., eds.: Proceedings of the 10th International Conference on Inductive Logic Programming. Volume 1866 of LNAI., Springer (2000) 40–59
10. Nienhuys-Cheng, S., de Wolf, R.: Foundations of Inductive Logic Programming. Volume 1228 of LNAI. Springer (1997)
11. Esposito, F., Fanizzi, N., Iannone, L., Semeraro, G.: Refinement of conceptual descriptions in $\mathcal{ALC}$ knowledge bases. Technical Report DL-2004-01, LACAM, Dipartimento di Informatica, Università degli Studi di Bari (2004)