A Refinement Operator for Description Logics

Liviu Badea[1] and **Shan-Hwei Nienhuys-Cheng**[2]

[1] AI Lab, National Institute for Research and Development in Informatics
8-10 Averescu Blvd., Bucharest, Romania.
e-mail: badea@ici.ro
[2] Erasmus University Rotterdam, H9-14, Post Box 1738
3000 DR, Rotterdam, The Netherlands.
e-mail: cheng@few.eur.nl

Abstract. While the problem of learning logic programs has been extensively studied in ILP, the problem of learning in description logics (DLs) has been tackled mostly by empirical means. Learning in DLs is however worthwhile, since both Horn logic and description logics are widely used knowledge representation formalisms, their expressive powers being incomparable (neither includes the other as a fragment). Unlike most approaches to learning in description logics, which provide bottom- up (and typically overly specific) *least* generalizations of the examples, this paper addresses learning in DLs using downward (and upward) refinement operators. Technically, we construct a *complete* and *proper* refinement operator for the $\mathcal{ALER}$ description logic (to avoid overfitting, we disallow disjunctions from the target DL). Although no *minimal* refinement operators exist for $\mathcal{ALER}$, we show that we can achieve minimality of all refinement steps, except the ones that introduce the $\bot$ concept. We additionally prove that complete refinement operators for $\mathcal{ALER}$ cannot be *locally finite* and suggest how this problem can be overcome by an MDL search heuristic. We also discuss the influence of the Open World Assumption (typically made in DLs) on example coverage.

1 Introduction

The field of machine learning has witnessed an evolution from ad-hoc specialized systems to increasingly more general algorithms and languages. This is not surprising since a learning algorithm aims at improving the behaviour of an *existing* system. And since early systems were quite diverse, the early learning systems were ad- hoc and thus hard to capture in a unified framework. Nevertheless, important progresses were made in the last decade towards learning in very general settings, such as first order logic. Inductive Logic Programming (ILP) deals with learning first order logic programs. Very recently the expressiveness of the target language was extended to prenex conjunctive normal forms [20] by allowing existential quantifiers in the language.

Description Logics (DL), on the other hand, are a different kind of knowledge representation language used for representing structural knowledge and concept hierarchies. They represent a function-free first order fragment allowing a variable-free syntax, which is considered to be important for reasons of readability (the readability of the specifications is an important issue in knowledge representation, where the expressiveness and tractability of the language are only considered together with the simplicity and understandability of the representations).

Description logics are subsets of first order logic containing only unary predicates (called *concepts* and representing sets of objects in the domain) and binary predicates (referred to as *roles* and representing binary relations between domain objects). A number of concept and role *constructors* can be used to express compound concept terms, which are variable-free representations of first-order descriptions.

The research in description logics has concentrated on (practical) algorithms and precise complexity results [9] for testing concept subsumption (inclusion) and consistency, as well as checking the membership of objects (*individuals*) in concepts for various combinations of concept and role constructors. These inference services are particularly useful for managing hierarchical knowledge (description logics usually provide automatic classification of concepts in a concept hierarchy).

Since the expressivities of Horn logic and description logics are incomparable [5], and since one of the main limitations of Horn rules is their inability to model and reason about value restrictions in domain with a rich hierarchical structure, there have been several attempts at combining description logics and (function-free) Horn rules (for example AL-log [10], CARIN [14, 15]). Reasoning in such combined languages is in general harder than in the separate languages [15].

While deduction in description logics has been thoroughly investigated and also while learning Horn rules has already reached a mature state (in the field of Inductive Logic Programming), learning DL descriptions from examples has been approached mostly by heuristic means [7, 8, 12]. For example, LCSLEARN [8] is a bottom-up learning algorithm using least common subsumers (LCS) as generalizations of concepts. It's disjunctive version, LCSLEARNDISJ, is similar to the ILP system GOLEM since LCSs play the role of least general generalizers (LGGs). Bottom-up ILP systems like GOLEM proved quite successful in certain applications (such as determining protein secondary structure), but they usually produce overly specific hypotheses. In Inductive Logic Programming, this drawback was eliminated by top-down systems like FOIL and Progol [16], which use *downward* refinement operators for exploring the space of hypotheses.

This paper aims at going beyond simple LCS-based learning in DLs by constructing complete upward and downward refinement operators for the description logic $\mathcal{ALER}$. This complete refinement operator is used by a more sophisticated learning algorithm to induce DL descriptions from examples.

Since description logics are a fragment of first-order logic, we could in principle translate DL expressions into prenex conjunctive form (PCNF) and use the PCNF-refinement operator defined by [20] to refine the PCNF encoding of DL expressions.[3]

Example 1. We could obtain the DL definition $Influencial \leftarrow \forall Friend.Influencial$ by the following squence of PCNF refinement steps:
Adding literals:

$\square \rightarrow \forall x I(x)$

$\rightarrow \forall x \forall y \forall z I(x) \vee F(y,z)$

$\rightarrow \forall x \forall y \forall z \forall u (I(x) \vee F(y,z)) \wedge I(u)$

[3] We need a refinement operator for PCNFs rather than universally quantified clauses because description logic expressions containing existential restrictions introduce existential quantifiers into their first- order encodings.

$\rightarrow \forall x \forall y \forall z \forall u \forall v (I(x) \vee F(y,z)) \wedge (I(u) \vee \neg I(v))$

Unifications:

$\rightarrow \forall x \forall z \forall u \forall v (I(x) \vee F(x,z)) \wedge (I(u) \vee \neg I(v))$

$\rightarrow \forall x \forall z \forall v (I(x) \vee F(x,z)) \wedge (I(x) \vee \neg I(v))$

$\rightarrow \forall x \forall z (I(x) \vee F(x,z)) \wedge (I(x) \vee \neg I(z))$

Substitutions:

$\rightarrow \forall x (I(x) \vee F(x,c)) \wedge (I(x) \vee \neg I(c))$, where c is a new constant

e-substitution:

$\rightarrow \exists y \forall x (I(x) \vee F(x,y)) \wedge (I(x) \vee \neg I(y))$

eu-substitution:

$\rightarrow \forall x \exists y (I(x) \vee F(x,y)) \wedge (I(x) \vee \neg I(y))$

However, this straightforward approach has two disadvantages:

- The conversion from DL to PCNF can lead to an (exponential) blow-up of the formulae. Converting everything to clausal form may spoil the structure of the initial DL description and the conversion of the refinement back to DL may affect the readability of the result.
- In fact, since DL descriptions are coarser grained than FOL formulae, some PCNF refinements may not even have a DL counterpart. More precisely, if ρ_{DL} is a DL-refinement operator and ρ a PCNF-refinement operator, then for every $D \in \rho_{DL}(C)$ we have $PCNF(D) \in \rho^*(PCNF(C))$, where $PCNF(C)$ is the PCNF encoding of the DL expression C. However, not every $D' \in \rho(PCNF(C))$ is the PCNF counterpart of a DL formula D: $D' = PCNF(D)$. Worse still, there can be arbitrarily long PCNF refinement chains (of some DL concept) that have no DL correspondence at all. Apparently, the PCNF refinement steps are too fine grained.

To circumvent these problems, we need to develop a refinement operator working directly on DL formulae. But directly refining DL formulae has an even deeper justification than just convenience. In fact, the hypotheses language (in our case a DL) plays the role of *learning bias*, which determines the granularity of the generalization steps. A very fine grained language (like FOL or PCNF) may be unsuitable for generalizing coarser grained DL descriptions.

2 The learning problem in Description Logics

In Description Logics, concepts represent classes of objects in the domain of interest, while roles represent binary relations in the same domain.

Complex concepts ($C, D, \ldots$) and roles ($R, Q, S, \ldots$) in the $\mathcal{ALER}$ description logic can be built from atomic concepts (A) and primitive roles (P) using the following concept and the role constructors:

Concept constructor	Syntax	Interpretation
top concept	$\top$	Δ
bottom concept	$\bot$	$\emptyset$
negation of atoms	$\neg A$	$\Delta \setminus A^{\mathcal{I}}$
concept conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
value restriction	$\forall R.C$	$\{x \in \Delta \mid \forall y.(x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
existential restriction	$\exists R.C$	$\{x \in \Delta \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$

Role constructor	Syntax	Interpretation
role conjunction	$R_1 \sqcap R_2$	$R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$

An interpretation $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ consists of a non-empty set Δ (the interpretation domain) and an interpretation function $\cdot^{\mathcal{I}}$ which maps concepts to subsets of Δ and roles to subsets of $\Delta \times \Delta$ according to the relationships in the table above.

A knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ has two components: a Tbox (terminology) $\mathcal{T}$ and Abox (assertional component) $\mathcal{A}$. A terminology $\mathcal{T}$ contains a set of definitions (terminological axioms) as below, representing intensional knowledge about classes of objects:

Definition of A	Syntax	Interpretation
sufficient	$A \leftarrow C$	$A^{\mathcal{I}} \supseteq C^{\mathcal{I}}$
necessary	$A \rightarrow C$	$A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$
necessary and sufficient	$A = C$	$A^{\mathcal{I}} = C^{\mathcal{I}}$

The Abox $\mathcal{A}$ contains extensional information in the form of membership assertions stating that specific individuals are instances of certain concepts or are involved in relations with other individuals as tuples of certain roles.

Assertion	Syntax	Interpretation
concept instance	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role tuple	$R(a,b)$	$(a,b)^{\mathcal{I}} \in R^{\mathcal{I}}$

An interpretation satisfies a (terminological or assertional) axiom iff the conditions in the tables above hold. (The interpretation function maps individuals, such as a and b, to domain elements such that the *unique names assumption* is satisfied: $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ if $a \neq b$.)

An interpretation that satisfies all the axioms of the knowledge base $\mathcal{K}$ is a *model* of $\mathcal{K}$.

Description logics represent a first-order logic fragment written in a variable-free syntax. The first-order encoding of $\mathcal{ALER}$ descriptions is given below.

C	$C(x)$
$\top$	$true$
$\bot$	$false$
$\neg A$	$\neg A(x)$
$C_1 \sqcap C_2$	$C_1(x) \wedge C_2(x)$
$\forall R.C$	$\forall y.R(x,y) \rightarrow C(y)$
$\exists R.C$	$\exists y.R(x,y) \wedge C(y)$

R	$R(x,y)$
$R_1 \sqcap R_2$	$R_1(x,y) \wedge R_2(x,y)$

Definition	FOL encoding
$A \leftarrow C$	$\forall x.\ A(x) \leftarrow C(x)$
$A \rightarrow C$	$\forall x.\ A(x) \rightarrow C(x)$
$A = C$	$\forall x.\ A(x) \leftrightarrow C(x)$

Assertion	FOL encoding
$C(a)$	$C(a)$
$R(a,b)$	$R(a,b)$

Description logics trade expressive power for more efficient (dedicated) inference services, as well as for an increased readability. Unlike FOL reasoners, DLs provide *complete* decision algorithms (which are guaranteed to terminate and whose complexity is tightly controlled) for the following (deductive) reasoning services:

- *KB-satisfiability*: $\mathcal{K}$ is satisfiable iff it admits a model.
- *concept satisfiability*: C is satisfiable w.r.t. $\mathcal{K}$ iff $\mathcal{K}$ admits a model $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$.
- *concept equivalence*: C and D are equivalent w.r.t. $\mathcal{K}$, $\mathcal{K} \models C \equiv D$, iff $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$.
- *concept subsumption*: C is subsumed[4] by D w.r.t. $\mathcal{K}$, $\mathcal{K} \models C \rightarrow D$, iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$. (We sometimes also write $C \sqsubseteq D$ instead of $C \rightarrow D$ to emphasize that subsumption is a generality order.)
  C is *strictly subsumed* by D, $C \sqsubset D$ iff $C \sqsubseteq D$ and $C \not\equiv D$.
- *instance checking*: a is an instance of C, $\mathcal{K} \models C(a)$, iff the assertion $C(a)$ is satisfied in every model of $\mathcal{K}$.

All the above *deductive* inference services can be reduced to $\mathcal{ALCR}$ KB-satisfiability [6].

Typically, terminological axioms have been used in DLs mainly for stating constraints (in the form of necessary definitions) on concepts. In our learning framework however, we also need sufficient definitions for classifying individuals as being instances of a given concept. Although theoretical work has addressed the issue of very general terminological axioms (such as general inclusions [6]), most existing DL implementations do not allow sufficient definitions in the TBox. However, we can "simulate" a set of sufficient definitions $\{A \leftarrow C_1, \ldots, A \leftarrow C_n\}$ in a DL with disjunctions by using the necessary and sufficient definition $A = C_1 \sqcup \ldots \sqcup C_n \sqcup A'$, where A' is a new concept name.

In this paper we aim at endowing DLs with *inductive* inference services as well. The *learning problem* in DLs can be stated as follows.

Definition 1. Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL knowledge base. A subset $\mathcal{A}' \subseteq \mathcal{A}$ of the assertions $\mathcal{A}$ can be viewed as (ground) *examples* of the target concept A:

$$\mathcal{A}' = \{A(a_1), A(a_2), \ldots, \neg A(b_1), \neg A(b_2), \ldots\}.$$

The DL learning problem consists in inducing a set of concept definitions for A: $\mathcal{T}'' = \{A \leftarrow C_1, A \leftarrow C_2, \ldots\}$ that covers the examples $\mathcal{A}'$, i.e.

$$\langle \mathcal{T} \cup \mathcal{T}'', \mathcal{A} \setminus \mathcal{A}' \rangle \models \mathcal{A}'.$$

In other words, we replace the specific examples $\mathcal{A}'$ from the Abox with more general Tbox definitions $\mathcal{T}''$. Note that, after learning, the knowledge base will be $\langle \mathcal{T} \cup \mathcal{T}'', \mathcal{A} \rangle$, which is equivalent with $\langle \mathcal{T} \cup \mathcal{T}'', \mathcal{A} \setminus \mathcal{A}' \rangle$, since the latter is supposed to cover the examples $\mathcal{A}'$.

Alternatively, we could consider a more general setting in which Tbox definitions of A can also serve as examples. In this case, the examples are a knowledge base $\langle \mathcal{T}', \mathcal{A}' \rangle$ such that

$\mathcal{T}' = \{A \leftarrow D_1, A \leftarrow D_2, \ldots\} \subseteq \mathcal{T}$

$\mathcal{A}' = \{A(a_1), A(a_2), \ldots, \neg A(b_1), \neg A(b_2), \ldots\} \subseteq \mathcal{A}$

and the learning problem consists in inducing a set of concept definitions for A $\mathcal{T}'' = \{A \leftarrow C_1, A \leftarrow C_2, \ldots\}$ that cover the examples $\langle \mathcal{T}', \mathcal{A}' \rangle$, i.e.

$$\langle (\mathcal{T} \setminus \mathcal{T}') \cup \mathcal{T}'', \mathcal{A} \setminus \mathcal{A}' \rangle \models \langle \mathcal{T}', \mathcal{A}' \rangle.$$

[4] Concept subsumption in description logics should not be confused with clause subsumption in ILP and Automated Reasoning.

(The knowledge base after learning $\langle \mathcal{T} \cup \mathcal{T}'', \mathcal{A} \rangle$ is equivalent with $\langle (\mathcal{T} \setminus \mathcal{T}') \cup \mathcal{T}'', \mathcal{A} \backslash \mathcal{A}' \rangle$.) Note that, in this setting, $\langle \mathcal{T} \backslash \mathcal{T}', \mathcal{A} \backslash \mathcal{A}' \rangle$ plays the role of background knowledge.

The learning problem thus formulated is very similar to the standard setting of Inductive Logic Programming. The main differences consist in the different expressivities of the target hypotheses spaces and the *Open World Assumption* (OWA) adopted by DLs (as opposed to the Closed World Assumption usually made in (Inductive) Logic Programming).

3 $\mathcal{ALER}$ refinement operators

While Inductive Logic Programming (ILP) systems learn logic programs from examples, a DL-learning system should learn DL descriptions from Abox instances. Both types of learning systems traverse large spaces of hypotheses in an attempt to come up with an optimal consistent hypothesis as fast as possible. Various heuristics can be used to guide this search, for instance based on information gain, MDL [16], etc. A simple search algorithm (even a complete and non-redundant one) would not do, unless it allows for a flexible traversal of the search space, based on an external heuristic. Refinement operators allow us to decouple the heuristic from the search algorithm. Downward (upward) refinement operators construct specializations (generalizations) of hypotheses and are usable in a top-down (respectively bottom-up) search of the space of hypotheses.

Definition 2. A *downward (upward) refinement operator* is a mapping ρ from hypotheses to sets of hypotheses (called refinements) which produces only specializations (generalizations), i.e. $H' \in \rho(H)$ implies $H \models H'$ (respectively $H' \models H$). (We shall sometimes also write $H \rightsquigarrow H'$ instead of $H' \in \rho(H)$.)

If the hypotheses are *sufficient definitions*, then $A \leftarrow C_1$ is more general than (entails) $A \leftarrow C_2$ iff C_1 subsumes (is more general than) C_2 ($C_1 \sqsupseteq C_2$).

In ILP the *least general generalization (lgg)* of two clauses is the least general *clause* that subsumes both. Note that although such an *lgg* is more general than conjunction, taking the *lgg* is justified by the fact that the conjunction of clauses is not a clause and would only overfit the input clauses (i.e. wouldn't perform any generalization leap).

Similarly with ILP, we define the *lgg* of two sufficient definitions as

$$lgg(A \leftarrow C_1, A \leftarrow C_2) = A \leftarrow lcs(C_1, C_2),$$

where $lcs(C_1, C_2)$ is the *least common subsumer* [7] of the DL concepts C_1 and C_2. In order to avoid overfitting and allow generalization leaps, the DL used as a target language should not provide concept disjunctions[5], since the *lcs* would otherwise reduce to concept disjunction.

For hypotheses representing *necessary definitions*, $A \rightarrow C_1$ is more general than (entails) $A \rightarrow C_2$ iff C_1 is subsumed by (is more specific than) C_2 ($C_1 \sqsubseteq C_2$). The *lgg* of two necessary definitions reduces to concept conjunction

$$lgg(A \rightarrow C_1, A \rightarrow C_2) = A \rightarrow C_1 \sqcap C_2,$$

[5] this justifies our choice of the target language $\mathcal{ALER}$ instead of the more expressive $\mathcal{ALCR}$, which allows concept disjunctions.

while their *most general specialization (mgs)* is

$$mgs(A \to C_1, A \to C_2) = A \to lcs(C_1, C_2).$$

While sufficient definitions ($A \leftarrow C_{suff}$) represent lower bounds on the target concept A, necessary definitions ($A \to C_{nec}$) represent upper bounds on A, i.e. $C_{suff} \sqsubseteq A \sqsubseteq C_{nec}$.

Note that a downward refinement operator on concept descriptions C (going from $\top$ to $\bot$) induces a downward refinement operator on sufficient definitions $A \leftarrow C$ and an upward refinement operator on necessary definitions $A \to C$.

But unlike necessary definitions $A \to C_1, \ldots, A \to C_n$, whose conjunction can be expressed as a single necessary definition $A \to C_1 \sqcap \ldots \sqcap C_n$, sufficient definitions like $A \leftarrow C_1, \ldots, A \leftarrow C_n$ cannot be expressed as a single sufficient definition unless the language allows concept disjunction: $A \leftarrow C_1 \sqcup \ldots \sqcup C_n$.[6]

In the following, we construct a *complete* downward refinement operator for $\mathcal{ALER}$ concepts.

Definition 3. A downward refinement operator ρ on a set of concepts ordered by the subsumption relationship $\sqsupseteq$ is called

- *(locally) finite* iff $\rho(C)$ is finite for every hypothesis C.
- *complete* iff for all C and D, if C is strictly more general than D ($C \sqsupset D$), then $\exists E \in \rho^*(C)$ such that $E \equiv D$.
- *weakly complete* iff $\rho^*(\top) =$ the entire set of hypotheses.
- *redundant* iff there exists a refinement chain[7] from C_1 to D not going through C_2 and a refinement chain from C_2 to D not going through C_1.
- *minimal* iff for all C, $\rho(C)$ contains only downward covers[8] and all its elements are incomparable.
- *proper* iff for all C and D, $D \in \rho(C)$ entails $D \sqsubset C$ (or, equivalently, $D \not\equiv C$).

(For defining the corresponding properties of *upward* refinement operators, we simply need to replace the generality order $\sqsupseteq$ by its dual $\sqsubseteq$.)

We first construct a complete but *non-minimal* (and thus highly redundant[9]) refinement operator for which the completeness proof is rather simple. Subsequently, we will modify this operator (while preserving its completeness) to reduce its non-minimality.

---

[6] Of course, such necessary definitions could be approximated by $A \leftarrow lcs(C_1, \ldots, C_n)$, but this is more general than the conjunction of the original definitions.

[7] A *refinement chain* from C to D is a sequence $C_0, C_1, \ldots, C_n$ of hypotheses such that $C = C_0, C_1 \in \rho(C_0), C_2 \in \rho(C_1), \ldots, C_n \in \rho(C_{n-1}), D = C_n$. Such a refinement chain does not 'go through' E iff $E \neq C_i$ for $i = 0, \ldots, n$.

[8] D is a *downward cover* of C iff C is more general than D ($C \sqsupset D$) and no E satisfies $C \sqsupset E \sqsupset D$.

[9] There are two possible sources of *redundancy* in a refinement operator:
– non-minimal refinement steps, and
– the possibility of reaching a hypothesis from two different incomparable hypotheses.
Here we refer to the first type of redundancy, which can be eliminated by disallowing non-minimal steps. (The second type of redundancy can be eliminated using the methods from [3, 4].)

The refinement operator ρ_0 is given by the following refinement rules (recall that $C \rightsquigarrow D$ means $D \in \rho_0(C)$, which entails the fact that C subsumes D, $C \sqsupseteq D$):

Refinement rules of ρ_0

$[Lit]$ $C \rightsquigarrow C \sqcap L$ with L a *DL-literal* (to be defined below)
$[\exists C]$ $C \sqcap \exists R.C_1 \rightsquigarrow C \sqcap \exists R.C_2$ if $C_1 \rightsquigarrow C_2$
$[\exists R]$ $C \sqcap \exists R_1.D \rightsquigarrow C \sqcap \exists R_2.D$ if $R_1 \rightsquigarrow R_2$
$[\exists\exists]$ $C \sqcap \exists R_1.C_1 \sqcap \exists R_2.C_2 \rightsquigarrow C \sqcap \exists (R_1 \sqcap R_2).(C_1 \sqcap C_2)$
$[\forall C]$ $C \sqcap \forall R.C_1 \rightsquigarrow C \sqcap \forall R.C_2$ if $C_1 \rightsquigarrow C_2$
$[\forall R]$ $C \sqcap \forall R_1.D \rightsquigarrow C \sqcap \forall R_2.D$ if $R_2 \rightsquigarrow R_1$

$[PR]$ $R \rightsquigarrow R \sqcap P$ with P a primitive role.

The refinement rules above apply to concepts in $\mathcal{ALER}$-*normal form*, which can be obtained for a concept by applying the following identities as rewrite rules[10] left-to-right until they are no longer applicable:

$[\forall\forall]$ $\forall R.C \sqcap \forall R.D = \forall R.(C \sqcap D)$
$[\forall\forall R]$ $\forall R.C \sqcap \forall Q.D = \forall R.(C \sqcap D) \sqcap \forall Q.D$ if $R \sqsubset Q$
$[\exists\forall]$ $\exists R.C \sqcap \forall Q.D = \exists R.(C \sqcap D) \sqcap \forall Q.D$ if $R \sqsubseteq Q$
$\forall R.\top = \top$ $\exists R.\bot = \bot$
$C \sqcap \neg C = \bot$ $C \sqcap \top = C$ $C \sqcap \bot = \bot$

For example, the normal form of $\forall(P_1 \sqcap P_2).A_1 \sqcap \forall P_1.\neg A_1 \sqcap \forall P_1.A_2 \sqcap \exists P_1.A_3$ is $\forall(P_1 \sqcap P_2).\bot \sqcap \forall P_1.(\neg A_1 \sqcap A_2) \sqcap \exists P_1.(\neg A_1 \sqcap A_2 \sqcap A_3)$.

Definition 4. In the refinement rule $[Lit]$, a *DL-literal* is either

- an atom (A), the negation of an atom ($\neg A$),
- an existential restriction $\exists P.\top$ for a primitive role P, or
- a value restriction $\forall \prod_{i=1}^{n} P_i.L'$ with L' a *DL-literal*, where $\{P_1, .., P_n\}$ is the set of all primitive roles occurring in the knowledge base.

An example of a ρ_0-chain: $\top \overset{[Lit]}{\rightsquigarrow} A_1 \overset{[Lit]}{\rightsquigarrow} A_1 \sqcap \forall(P_1 \sqcap P_2).A_2 \overset{[Lit]}{\rightsquigarrow} A_1 \sqcap \forall(P_1 \sqcap P_2).A_2 \sqcap \exists P_2.\top \overset{[\exists C]}{\rightsquigarrow} A_1 \sqcap \forall(P_1 \sqcap P_2).A_2 \sqcap \exists P_2.A_3 \overset{[\forall R]}{\rightsquigarrow} A_1 \sqcap \forall P_1.A_2 \sqcap \exists P_2.A_3 \overset{[Lit]}{\rightsquigarrow} A_1 \sqcap \forall P_1.A_2 \sqcap \exists P_2.A_3 \sqcap \exists P_2.\top \overset{[\exists C]}{\rightsquigarrow} A_1 \sqcap \forall P_1.A_2 \sqcap \exists P_2.A_3 \sqcap \exists P_2.A_4 \overset{[\exists\exists]}{\rightsquigarrow} A_1 \sqcap \forall P_1.A_2 \sqcap \exists P_2.(A_3 \sqcap A_4)$.

The above definition of DL-literal can be explained as follows. The addition of new *"DL-literals"* in the $[Lit]$ rule should involve not just atoms and negations of atoms (i.e. ordinary literals), but also existential and value restrictions. For minimality, these have to be *most general* w.r.t. the concept to be refined C (as well as *non-redundant*, if possible).

The *most general existential restrictions* take the form $\exists P.\top$ for a primitive role P. But if P already occurs in some other existential restriction on the "top level" of C (i.e. $C = C' \sqcap \exists R_1.C_1$ such that $P \sqsupseteq R_1$, or, in other words, $R_1 = R_1' \sqcap P$), then $\exists P.\top$ is redundant w.r.t. C. This is due to the following result.

[10] Under associativity, commutativity and idempotence of $\sqcap$.

Proposition 5. *$\exists R_1.C_1 \sqsubseteq \exists R_2.C_2$ if $R_1 \sqsubseteq R_2$ and $C_1 \sqsubseteq C_2$.*

Consequently, the restriction to be added $\exists R_2.C_2$ is redundant w.r.t. some other existential restriction $\exists R_1.C_1$ (i.e. $\exists R_1.C_1 \sqcap \exists R_2.C_2 \equiv \exists R_1.C_1$) if $C_1 \sqsubseteq C_2$ and $R_1 \sqsubseteq R_2$.

But disallowing the addition of $\exists P.\top$ to C in cases in which some $\exists R_1.C_1$ with $R_1 \sqsubseteq P$ already occurs in C (which would ensure the *properness* of ρ_0) would unfortunately also lead to the *incompleteness* of ρ_0. For example, it would be impossible to reach $\exists P.A_1 \sqcap \exists P.A_2$ as a refinement of $\exists P.A_1$, because the first step in the following chain of refinements[11]:

$$\exists P.A_1 \overset{[Lit]}{\rightsquigarrow} \exists P.A_1 \sqcap \exists P.\top \overset{[\exists C]}{\rightsquigarrow} \exists P.A_1 \sqcap \exists P.A_2$$

would fail, due to the redundancy of $\exists P.\top$.

$\exists P.\top$ is redundant because it is too general. Maybe we could try to directly add something more specific, but non-redundant (like $\exists P.A_2$ in the example above). However, determining the most general *non-redundant* existential restrictions is complicated. We will therefore allow the refinement operator ρ to be improper (i.e. produce refinements $D \in \rho(C)$ that are equivalent to C[12]), but – in order to obtain proper refinements – we will successively apply ρ until a strict refinement (i.e. some $D \sqsubset C$) is produced. The resulting refinement operator ρ^{cl} (the "closure" of ρ) will be proper, by construction. More precisely:

Definition 6. $D \in \rho^{cl}(C)$ iff there exists a refinement chain of ρ:

$$\boxed{C} \overset{\rho}{\rightsquigarrow} C_1 \overset{\rho}{\rightsquigarrow} C_2 \overset{\rho}{\rightsquigarrow} \ldots \overset{\rho}{\rightsquigarrow} \boxed{C_n = D},$$

such that $C_i \equiv C$ for $i = 1, \ldots, n-1$ and $C_n \sqsubset C$.

In the above-mentioned example, we have the following refinement chain of ρ_0:

$$C = \exists P.A_1 \overset{[Lit]}{\rightsquigarrow} C_1 = \exists P.A_1 \sqcap \exists P.\top \overset{[\exists C]}{\rightsquigarrow} C_2 = \exists P.A_1 \sqcap \exists P.A_2$$

for which $C \equiv C_1$ and $C \sqsupset C_2$. Therefore, $C_2 \in \rho_0^{cl}(C)$ is a one-step refinement of the "closure" ρ_0^{cl}.

For determining the *most general value restrictions*, we consider the dual of Proposition 5:

---

[11] which is the only chain that can lead from $\exists P.A_1$ to $\exists P.A_1 \sqcap \exists P.A_2$

[12] The specific syntactic form of D is important in this case. In order to preserve completeness, we *disallow* the use of the following *redundancy elimination rules* (which will be used only for simplifying the result returned to the user):

$[\forall red]$ $\forall R_1.C_1 \sqcap \forall R_2.C_2 = \forall R_1.C_1$ if $R_1 \sqsupseteq R_2$ *and* $C_1 \sqsubseteq C_2$
$[\exists red]$ $\exists R_1.C_1 \sqcap \exists R_2.C_2 = \exists R_1.C_1$ if $R_1 \sqsubseteq R_2$ and $C_1 \sqsubseteq C_2$

For example, their use would disallow obtaining $\exists P.A_1 \sqcap \exists P.A_2$ from $\exists P.A_1$:

$$\exists P.A_1 \overset{[Lit]}{\rightsquigarrow} \exists P.A_1 \sqcap \exists P.\top \overset{[\exists C]}{\rightsquigarrow} \exists P.A_1 \sqcap \exists P.A_2$$

because $\exists P.A_1 \sqcap \exists P.\top$ would be simplified by the $[\exists red]$ redundancy elimination rule to $\exists P.A_1$, thereby making the second step ($[\exists C]$) inapplicable.

Proposition 7. $\forall R_1.C_1 \sqsubseteq \forall R_2.C_2$ *if* $R_1 \sqsupseteq R_2$ and $C_1 \sqsubseteq C_2$.

(In fact, we can prove an even stronger result: W.r.t. an $\mathcal{ALER}$ knowledge base $\mathcal{K}$ with no necessary definitions, $\mathcal{K} \models \forall R_1.C_1 \sqsubseteq \forall R_2.C_2$ iff $\mathcal{K} \models R_1 \sqsupseteq R_2$ *and* $\mathcal{K} \models C_1 \sqsubseteq C_2$. Thus, the restriction to be added $\forall R_2.C_2$ is non-redundant w.r.t. $\forall R_1.C_1$ (i.e. $\forall R_1.C_1 \sqcap \forall R_2.C_2 \not\equiv \forall R_1.C_1$) iff $R_1 \not\sqsupseteq R_2$ or $C_1 \not\sqsubseteq C_2$.)

Formally, the most general value restrictions take the form $\forall R.\top$. But unfortunately, such value restrictions are redundant due to the identity $\forall R.\top = \top$. Less redundant value restrictions are $\forall R.L'$, where L' is a DL-literal. Note that R in $\forall R.L'$ cannot be just a primitive role P, since for example $\forall(P \sqcap R').L'$ is more general than (subsumes) $\forall P.L'$. With respect to R, the most general value restriction thus involves a conjunction of all primitive roles in the knowledge base $\forall \prod_{i=1}^{n} P_i.L'$, but unfortunately this is, in general, also redundant. As shown above, redundancy can be eliminated by considering the "closure" ρ_0^{cl} of ρ_0.

3.1 Reducing non-minimality

It is relatively easy to show that the refinement operator ρ_0 (as well as its closure ρ_0^{cl}) is *complete*. However, it is *non-minimal* (and thereby highly redundant), due to the following $\mathcal{ALER}$ relationships:

$$\exists R.C_1 \sqcap \exists R.C_2 \sqsupseteq \exists R.(C_1 \sqcap C_2), \tag{1}$$
$$\forall R.C_1 \sqcap \forall R.C_2 = \forall R.(C_1 \sqcap C_2). \tag{2}$$

- (1) suggests that, for reasons of minimality, we should not allow in $[\exists C]\ C_1$ inside $\exists R.C_1$ to be refined by literal additions ($[Lit]$) to $C_1 \sqcap L$:

  $$C \sqcap \exists R.C_1 \overset{[\exists C]}{\rightsquigarrow} C \sqcap \exists R.(C_1 \sqcap L),$$

  because this single-step refinement could also be obtained with the following sequence of smaller steps (the $[\exists\top]$ step is defined below):

  $$C \sqcap \exists R.C_1 \overset{[Lit]}{\rightsquigarrow} C \sqcap \exists R.C_1 \sqcap \exists R.\top \overset{[\exists\top]}{\rightsquigarrow} C \sqcap \exists R.C_1 \sqcap \exists R.L \overset{[\exists\exists]}{\rightsquigarrow} C \sqcap \exists R.(C_1 \sqcap L).$$

  In other words, instead of directly refining $\exists R.C_1$ to $\exists R.(C_1 \sqcap L)$, we first add $\exists R.L$, and then merge $\exists R.C_1$ and $\exists R.L$ to $\exists R.(C_1 \sqcap L)$ using $[\exists\exists]$ (the refinement step being justified by (1)).

- Similarly, (2) suggests that, for reasons of minimality, we should disallow in $[\forall C]\ C_1$ inside $\forall R.C_1$ to be refined by literal additions ($[Lit]$) to $C_1 \sqcap L$:

  $$C \sqcap \forall R.C_1 \overset{[\forall C]}{\rightsquigarrow} C \sqcap \forall R.(C_1 \sqcap L),$$

  because this single-step refinement could also be obtained with the following sequence of smaller steps:

  $$C \sqcap \forall R.C_1 \overset{[Lit]}{\rightsquigarrow} C \sqcap \forall R.C_1 \sqcap \forall(R \sqcap \ldots).L \overset{[\forall R]}{\rightsquigarrow} \ldots \overset{[\forall R]}{\rightsquigarrow} C \sqcap \forall R.C_1 \sqcap \forall R.L \overset{[\forall\forall]}{=} C \sqcap \forall R.(C_1 \sqcap L).$$

  (In the last step, we applied the simplification rule $[\forall\forall]$.)

- Thirdly, the relationship

$$\exists R_1.C \sqcap \exists R_2.C \sqsupseteq \exists (R_1 \sqcap R_2).C \tag{3}$$

suggests that, for reasons of minimality, we should also drop the $[\exists R]$ altogether. The rule $[\exists R]$ of ρ_0 is redundant since the single step

$$C \sqcap \exists R_1.D \overset{[\exists R]}{\rightsquigarrow} C \sqcap \exists (R_1 \sqcap P).D$$

can also be obtained in several steps, as follows:

$$C \sqcap \exists R_1.D \overset{[Lit]}{\rightsquigarrow} C \sqcap \exists R_1.D \sqcap \exists P.\top \overset{[\exists \top]}{\rightsquigarrow} \ldots \rightsquigarrow C \sqcap \exists R_1.D \sqcap \exists P.D \overset{[\exists\exists]}{\rightsquigarrow} C \sqcap \exists (R_1 \sqcap P).D.$$

Thus, instead of directly refining $\exists R_1.D$ to $\exists (R_1 \sqcap P).D$, we first add $\exists P.D$ and then merge $\exists R_1.D$ and $\exists P.D$ to $\exists (R_1 \sqcap P).D$ using $[\exists\exists]$ (the refinement step being justified by (3)).

- Finally, since the $[Lit]$ step of ρ_0 can add literals L that are complementary to an already existing literal from C, we can obtain inconsistent concepts as refinements of any C. Of course, although $C \rightsquigarrow \bot$ is a valid refinement step (because $C \sqsupseteq \bot$), it is not only non-minimal, but also useless if it is applied on the "top level"[13] of the concept to be refined. However, refining some *subconcept* (of the concept to be refined) to $\bot$ makes sense, for example $\forall R.C \rightsquigarrow \forall R.\bot$ ($\forall R.\bot$ being consistent!), although it is still non-minimal. Unfortunately, as we show below, all refinements $C_1 \rightsquigarrow C_2$ that introduce a new $\bot$ in (some subconcept of) C_2 are always non-minimal, so we have to make a *trade-off between the minimality and the completeness* of the refinement operator. If we want to preserve completeness, we need to allow refinement steps like $C \rightsquigarrow \bot$, if not on the "top level" of the concept to be refined (i.e. in the rule $[Lit]$) or in $[\exists C]$ (where allowing $C \sqcap \exists R_1.C \rightsquigarrow C \sqcap \exists R_1.\bot = \bot$ would lead to an inconsistency), then at least in $[\forall C]$, which has to be modified as follows:
  $[\forall C]$ $\quad C \sqcap \forall R.C_1 \rightsquigarrow C \sqcap \forall R.C_2$ where $C_1 \rightsquigarrow C_2$ **or** $C_2 = \bot$.
  However, besides this explicit introduction of $\bot$ in $\forall$ restrictions, we shall require refinements to be *consistent*. More formally, $D \in \rho(C)$ iff D is obtained as a refinement of C using the refinement rules below ($C \rightsquigarrow D$) and D is *consistent*.

The resulting refinement operator ρ presented below treats literal additions $[Lit]$ in a special manner (since $[Lit]$ is not allowed to be recursively used in $[\exists C]$ or $[\forall C]$ rules). We therefore let ρ' denote all the rules of ρ except $[Lit]$:

Refinement rules of ρ'

$[\exists \top]$ $C \sqcap \exists R.\top \rightsquigarrow C \sqcap \exists R.L$ $\quad$ with L a *DL-literal*

$[\exists C]$ $C \sqcap \exists R.C_1 \rightsquigarrow C \sqcap \exists R.C_2$ $\quad$ if $C_1 \overset{\rho'}{\rightsquigarrow} C_2$
$[\exists\exists]$ $C \sqcap \exists R_1.C_1 \sqcap \exists R_2.C_2 \rightsquigarrow C \sqcap \exists (R_1 \sqcap R_2).(C_1 \sqcap C_2)$

$[\forall C]$ $C \sqcap \forall R.C_1 \rightsquigarrow C \sqcap \forall R.C_2$ $\quad$ if $C_1 \overset{\rho'}{\rightsquigarrow} C_2$ or $C_2 = \bot$
$[\forall R]$ $C \sqcap \forall R_1.D \rightsquigarrow C \sqcap \forall R_2.D$ $\quad$ if $R_2 \rightsquigarrow R_1$

$[PR]$ $R \rightsquigarrow R \sqcap P$ $\quad$ with P a primitive role.

[13] i.e. to C, as opposed to its subconcepts.

In the following, we shall write $C \overset{\mathbf{Rule}}{\rightsquigarrow} D$ whenever D is obtained as a refinement of C using the refinement rule **Rule** (which can be $[\exists\top]$, $[\exists C]$, $[\exists\exists]$, $[\forall C]$, $[\forall R]$, or – in the case of ρ – also $[Lit]$). Moreover, we sometimes write $C \overset{\rho'}{\rightsquigarrow} D$ instead of $D \in \rho'(C)$ (denoting the fact that D is obtained as a refinement of C *without using the* $[Lit]$ *rule*).

The refinement rules of the complete refinement operator ρ are the refinement rules of ρ' together with the $[Lit]$ rule.

Refinement rules of ρ

$[\rho']$ refinement rules of ρ'
$[Lit]$ $C \rightsquigarrow C \sqcap L$ with L a *DL-literal* such that $C \sqcap L$ is consistent.

We recall that we have defined $D \in \rho(C)$ iff $C \rightsquigarrow D$ and D is consistent.

3.2 Properties of ρ

1. **Completeness.** Since ρ_0 is complete and the modification of ρ_0 to ρ preserves completeness, ρ will be *complete* too.
   Note that we are adding DL-literals either on the "top level" of the concept to be refined (using $[Lit]$), or inside $\exists R.\top$ restrictions (in $[\exists\top]$). Such literals can be "moved inside" $\forall \cdot \exists \cdot \forall \cdots$ chains by using the $[\exists\exists]$ refinement rule and the $[\forall\forall]$ rewrite rule.
2. **Properness.** Like in the case of ρ_0, ρ is *not proper* because the DL-literals of the form $\exists P.\top$ inserted can be redundant. For achieving properness, we should consider the closure ρ^{cl}, rather than simplifying these redundancies (which would affect completeness).
3. **Minimality.** By construction, ρ has less non-minimal steps than ρ_0. However, there exists a fundamental trade-off between *minimality* and *completeness* of $\mathcal{ALER}$ refinement operators in general (not just for ours).

   Proposition 8. *There exist no* minimal *and* complete $\mathcal{ALER}$ *refinement operators.*[14]

   Example 2. The following infinite chain of minimal refinements between $\forall P.A$ and $\forall P.\bot$

$$\begin{aligned} C &= \forall P.A \ \sqsupset \\ C_1 &= \forall P.(A \sqcap \forall P.A) \ \sqsupset \\ C_2 &= \forall P.(A \sqcap \forall P.(A \sqcap \forall P.A)) \ \sqsupset \\ C_3 &= \forall P.(A \sqcap \forall P.(A \sqcap \forall P.(A \sqcap \forall P.A))) \ \sqsupset \ldots \sqsupset \\ C_\infty &= \forall P.\bot \end{aligned}$$

[14] There cannot exist a *minimal* refinement step $C \rightsquigarrow \forall P.\bot$, since there exists a sufficiently large n (for example, larger than the size of C) such that $C \sqsupset C \sqcap \underbrace{\forall P. \ldots \forall P.}_{n} A \sqsupset \forall P.\bot$.

shows that a *minimal* refinement operator will not be able to reach $\forall P.\bot$ from $\forall P.A$ in a *finite* number of steps, thereby being *incomplete*.
On the other hand, if we insist on *completeness*, we should allow $\forall P.\bot$ as a refinement of some C_i, thereby making the refinement operator *non-minimal*.

Our refinement operator allows $\forall P.\bot$ as a refinement of any C_i in the example above. It is therefore non-minimal. However, we conjecture that although there exist no minimal and complete refinement operators for $\mathcal{ALER}$, there exist refinement operators all of whose steps $C \rightsquigarrow D$ are minimal, *except for the steps involving the introduction of* $\bot$ *in some subconcept of* D (like in the $C_2 = \bot$ case of the $[\forall C]$ rule of our refinement operator ρ). This suggests that our ρ is one of the best refinement operators one can hope for.

4. **Local finiteness.** The following example shows that there can be no locally finite and complete $\mathcal{ALER}$ refinement operators.

   Example 3. A_1 admits the following infinite set of minimal direct refinements:

   $$\{A_1 \sqcap \forall P.A, \quad A_1 \sqcap \forall P.\forall P.A, \quad A_1 \sqcap \forall P.\forall P.\forall P.A, \quad \ldots\}.$$

   Therefore, since ρ is complete, it will not be locally finite either. Apparently, this seems to be a significant problem. However, as the example above suggests, the infinite set of minimal direct refinements of some concept C involves increasingly longer concepts D, which will be immediately discarded by a refinement heuristic taking into account the size of hypotheses:

   $$f(H) = pos(H) - neg(H) - size(H)$$

   (where $pos(H)$ and $neg(H)$ are the number of positive/negative examples covered by the hypothesis H.)
   Note that the lack of local finiteness and respectively minimality of a complete $\mathcal{ALER}$ refinement operator seem to be related. (They seem to involve value restrictions and $\bot$ in an essential way[15]). This situation also seems to be related to the non-existence of the Most Specific Concept (MSC) of individuals in some concept languages, such as $\mathcal{ALN}$ [1].

4 Testing example coverage

Although both Horn-clause logic programming (LP) and description logics (DL) are fragments of first order logic and are therefore similar in certain respects, there are also some significant differences.

- DLs make the *Open World Assumption (OWA)*, while LP makes the *Closed World Assumption (CWA)*.

[15] The situation is unlike in ILP, where such problems do not occur (obviously, because value restrictions cannot be expressed in Logic Programming).
In $\mathcal{ALER}$, on the other hand, we do not have infinite ascending chains [18] because $\mathcal{ALER}$ does not allow the construction of cyclic descriptions (such as $R(X_1, X_2), R(X_2, X_3), R(X_3, X_1)$, for example).

– DL definitions like $A \leftarrow \forall R.C$ and $A \rightarrow \exists R.C$ involve existentially quantified variables and thus cannot be expressed in pure LP. Using the meta-predicate *forall*, we could approximate $A \leftarrow \forall R.C$ as $A(X) \leftarrow forall(R(X,Y), C(Y))$. But while the former definition is interpreted w.r.t. the OWA, the latter is interpreted w.r.t. the CWA, which makes it closer to $A \leftarrow \forall KR.C$. where K is an epistemic operator as in [11]. Also, while DLs provide inference services (like subsumption checking) to reason about such descriptions, LP systems with the meta-predicate *forall* can only answer queries, but not reason about such descriptions.

Although the OWA is sometimes preferable to the CWA[16], the OWA is a problem when testing that a definition, for example $A \leftarrow \forall R.C$, covers a given example, for instance $A(a_i)$. Examples are unavoidably *incomplete*. Even if all the *known* R-fillers of a_i from the Abox verify C:

$$\mathcal{A} = \{R(a_i, b_{i1}), C(b_{i1}), \ldots, R(a_i, b_{in}), C(b_{in})\}$$

this doesn't mean that a_i verifies $\forall R.C$[17], so the antecedent $\forall R.C$ of $A \leftarrow \forall R.C$ will never be satisfied by an example a_i (unless explicitly stated in the KB). However, a_i will verify $\forall KR.C$ because all the *known* R-fillers of a_i verify C, so the definition $A \leftarrow \forall KR.C$ covers the example $A(a_i)$, as expected.

Thus, when checking example coverage. we need to "close" the roles (for example, by replacing R with KR, or, equivalently, assuming that the known fillers are all the fillers).

4.1 Example coverage for sufficient definitions

Definition 9. In the case of a DL knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$, for which $\mathcal{A}' = \{A(a_1), \ldots, A(a_p), \neg A(a_{p+1}), \ldots, \neg A(a_{p+n})\} \subseteq \mathcal{A}$ are considered (positive and negative) examples, we say that the sufficient definition $A \leftarrow C$ *covers* the (positive or negative) example a_i iff

$$cl\langle \mathcal{T} \cup \{A \leftarrow C\}, \mathcal{A} \setminus \mathcal{A}' \rangle \models A(a_i),$$

which is equivalent with the inconsistency of

$$cl\langle \mathcal{T} \cup \{A \leftarrow C\}, (\mathcal{A} \setminus \mathcal{A}') \cup \{\neg A(a_i)\} \rangle,$$

where $cl\langle \mathcal{T}, \mathcal{A} \rangle$ denotes the role-closure of the knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$ (which amounts to replacing roles R with KR).

Example 4. Consider the following knowledge base $\langle \mathcal{T}, \mathcal{A} \rangle$ with an empty Tbox ($\mathcal{T} = \emptyset$) and the Abox[18]

$$\begin{aligned} \mathcal{A} = \{ & IP(j), R(j), F(j, j_1), I(j_1), F(j, j_2), I(j_2), \\ & \neg IP(f), R(f), \\ & \neg IP(m), R(m), F(m, m_1), I(m_1), F(m, m_2), \\ & \neg IP(h), F(h, h_1), I(h_1), F(h, h_2), I(h_2)\}, \end{aligned}$$

[16] For example, when expressing constraints on role fillers using value restrictions.

[17] because of the OWA, there could exist, in principle, a yet unknown R-filler of a_i not verifying C.

[18] Where the individuals *j, m, f, h* stand for *John, Mary, Fred, Helen*, while the atomic concepts *IP, R, I* stand for *Influencial_Person, Rich, Influencial* and the primitive role F for *Friend*.

where $\mathcal{A}' = \{IP(j), \neg IP(f), \neg IP(m), \neg IP(h)\}$ are considered as positive and negative examples.

The following sufficient definition covers all positive examples while avoiding all (not covering any) negative examples:

$$IP \leftarrow R \sqcap \forall F.I \sqcap \exists F.I \tag{4}$$

(4) covers the positive example $IP(j)$ because $\langle \{IP \leftarrow R \sqcap \forall KF.I \sqcap \exists KF.I\}, (\mathcal{A} \setminus \mathcal{A}') \cup \{\neg IP(j)\}\rangle$ is inconsistent, since $\neg R(j)$, $(\neg \forall KF.I)(j)$, $(\neg \exists KF.I)(j)$ are all inconsistent:

- $\neg R(j)$ because $R(j) \in \mathcal{A} \setminus \mathcal{A}'$
- $(\neg \forall KF.I)(j)$, i.e. $(\exists KF.\neg I)(j)$ because all the *known* F-fillers of j (namely j_1 and j_2) are I
- $(\neg \exists KF.I)(j)$, i.e. $(\forall KF.\neg I)(j)$ because there exists an F-filler of j (for example j_1) that is I.

Note that the more general definition $IP \leftarrow R \sqcap \forall F.I$ covers the negative example $\neg IP(f)$ because $\langle \{IP \leftarrow R \sqcap \forall KF.I\}, (\mathcal{A} \setminus \mathcal{A}') \cup \{\neg IP(f)\}\rangle$ is inconsistent, since $\neg R(f)$, $(\neg \forall KF.I)(f)$ are both inconsistent:

- $\neg R(f)$ because $R(f) \in \mathcal{A} \setminus \mathcal{A}'$
- $(\neg \forall KF.I)(f)$, i.e. $(\exists KF.\neg I)(f)$ because there exists no *known* F-filler of f.

And since the more specific (4) does not cover the negative example $\neg IP(f)$ (due to the consistency of $(\neg \exists KF.I)(f)$), we conclude that $\exists F.I$ discriminates between the positive example $IP(j)$ and the negative example $\neg IP(f)$ and is therefore necessary in (4).

$IP \leftarrow \forall F.I \sqcap \exists F.I$ (which is also more general than (4)) covers the negative example $\neg IP(h)$ because $\langle \{IP \leftarrow \forall KF.I \sqcap \exists KF.I\}, (\mathcal{A} \setminus \mathcal{A}') \cup \{\neg IP(h)\}\rangle$ is inconsistent, since $(\neg \forall KF.I)(h)$, $(\neg \exists KF.I)(h)$ are inconsistent:

- $(\neg \forall KF.I)(h)$, i.e. $(\exists KF.\neg I)(h)$ because all the *known* F-fillers of h (namely h_1 and h_2) are I
- $(\neg \exists KF.I)(h)$, i.e. $(\forall KF.\neg I)(h)$ because there exists an F-filler of h (for example h_1) that is I.

And since the more specific (4) does not cover the negative example $\neg IP(h)$ (due to the consistency of $\neg R(h)$), we conclude that R discriminates between the positive example $IP(j)$ and the negative example $\neg IP(h)$ and is therefore necessary in (4).

$IP \leftarrow R \sqcap \exists F.I$ (which is also more general than (4)) covers the negative example $\neg IP(m)$ because $\langle \{IP \leftarrow R \sqcap \exists KF.I\}, (\mathcal{A} \setminus \mathcal{A}') \cup \{\neg IP(m)\}\rangle$ is inconsistent, since $\neg R(m)$, $(\neg \exists KF.I)(m)$ are inconsistent:

- $\neg R(m)$ because $R(m) \in \mathcal{A} \setminus \mathcal{A}'$
- $(\neg \exists KF.I)(m)$, i.e. $(\forall KF.\neg I)(m)$ because there exists an F-filler of m (namely m_1) that is I.

And since the more specific (4) does not cover the negative example $\neg IP(m)$ (due to the consistency of $(\neg \forall KF.I)(m)$), we conclude that $\forall F.I$ discriminates between the positive example $IP(j)$ and the negative example $\neg IP(m)$ and is therefore necessary in (4).

The right-hand side of (4) is obtained by the refinement operator ρ from $\top$ by the following sequence of steps:

$C_0 = \top \overset{[Lit]}{\rightsquigarrow} C_1 = R \overset{[Lit]}{\rightsquigarrow} C_2 = R \sqcap \forall F.I \overset{[Lit]}{\rightsquigarrow} C_3 = R \sqcap \forall F.I \sqcap \exists F.\top \overset{[\exists\forall]}{=} R \sqcap \forall F.I \sqcap \exists F.I$.

All C_i above cover the positive example $IP(j)$. However, C_0 covers all 3 negative examples, C_1 only $\neg IP(f)$, $\neg IP(m)$, C_2 only $\neg IP(f)$, while C_3 avoids all negative examples and would be returned as a solution: $IP \leftarrow C_3$.

4.2 Verifying necessary definitions

While sufficient definitions can be used to classify individuals as instances of the target concept, necessary definitions impose constraints on the instances of the target concept. Roughly speaking, a necessary definition $A \rightarrow C$ is *verified* iff it is entailed by the knowledge base: $\langle \mathcal{T}, \mathcal{A} \rangle \models (A \rightarrow C)$, which can be reduced to the inconsistency of $A \sqcap \neg C$ w.r.t. $\langle \mathcal{T}, \mathcal{A} \rangle$, i.e. to the non-existence of an instance x of $A \sqcap \neg C$. Such an x could be either a_i[19], another *known* individual, or a *new* one. Since the examples a_i of A are unavoidably incomplete, $A \rightarrow C$ will not be *provable* for all imaginable instances x. Equivalently, $(A \sqcap \neg C)(x)$ will not (and need not) be inconsistent for any x. In fact, we need to prove $A \rightarrow C$ (or, equivalently, to check the inconsistency of $A \sqcap \neg C$) only for the *known* examples a_i of A. This amounts to proving $(KA) \rightarrow C$, i.e. to considering the closure of the target concept A. Since $A(a_i)$ holds anyway, we just need to prove $C(a_i)$ for all positive examples a_i of A. More precisely:

Definition 10. The necessary definition $A \rightarrow C$ is *verified* in $\langle \mathcal{T}, \mathcal{A} \rangle$ iff $\forall A(a_i) \in \mathcal{A}'$, $cl\langle \mathcal{T}, \mathcal{A} \cup \{\neg C(a_i)\} \rangle$ is inconsistent.

Note that the above definition can be considered to obey the so-called *'provability view'* of constraints. (Adopting the weaker *'consistency view'* may be too weak in our learning framework: assuming that a constraint is verified just because it is consistent with the examples may lead to the adoption of too strong – and thus unjustified – constraints.)

5 Learning in DLs using refinement operators

A top-down DL learning algorithm would simply refine a very general definition of the target concept, like $A \leftarrow \top$, using a downward refinement operator[20] until it covers no negative examples. (A heuristic maximizing the number of positive examples covered, while minimizing the size of the definitions as well as the number of negative examples covered can be used to guide the search.) If this first covering step still leaves some positive examples uncovered, then subsequent covering steps will be employed to learn additional definitions until either all positive examples are covered, or the learning process fails due to the impossibility to cover certain positive examples without also covering (some) negative examples.

[19] Assuming that the known positive examples of A are $\mathcal{A}' = \{A(a_1), \ldots, A(a_n)\}$.

[20] The inherent redundancy of the *complete* refinement operator ρ needs to be eliminated by converting it into a *weakly complete* operator. This can be done using the methods from [3, 4].

Note that our approach avoids the difficulties faced by bottom-up approaches, which need to compute the minimal Tbox generalizations $MSC(a_i)$ (called *most specific concepts* [8, 1]) of the Abox examples $A(a_i)$. Most existing bottom-up approaches (such as [8]) then use *least common subsumers (LCS)* [7, 8, 2] to generalize the MSC descriptions. Unfortunately, such approaches tend to produce overly specific concept definitions. On the other hand, by reverting the arrows in our downward refinement operator, we obtain an *upward* refinement operator for the description logic $\mathcal{ALER}$ which can be used to search the space of DL descriptions in a more flexible way than by using LCSs and also without being limited to considering only least generalizations.

6 Conclusions

This paper can be viewed as an attempt to apply ILP learning methods to description logics – a widely used knowledge representation formalism that is different from the language of Horn clauses, traditionally employed in ILP. Extending ILP learning methods to description logics is important for at least two reasons. First, description logics represent a new sort of *learning bias*, which necessitates a more sophisticated refinement operator (than typical ILP refinement operators). Second, description logics provide constructs, such as value restrictions, which cannot be expressed in Horn logic, thereby enhancing the expressivity of the language and making it more suitable for applications that involve rich hierarchical knowledge.

Since Horn logic (HL) and description logics (DLs) are complementary, developing refinement operators for DLs represents a significant step towards learning in an integrated framework comprising both HL and DL. Due to the differences in expressivities between DLs and HL, constructing DL and respectively HL refinement operators encounter different problems. Neither can be minimal (while preserving completeness), but for different reasons: in HL we can have infinite ascending chains, while in our $\mathcal{ALER}$ description logic we cannot, non-minimality being due to the interplay of value restrictions and $\bot$. We also discuss the impact of the Open World Assumption (usually employed in DLs) on learning and especially on example coverage, but this issue needs further investigation.

Since learning in even a very simple DL allowing for definitions of the form $C \leftarrow \exists R.(A_1 \sqcap \ldots \sqcap A_n)$ is NP-hard (Theorem 2 of [13]), learning in our framework will be NP-hard as well. However, we prefer to preserve a certain expressiveness of the language and plan to study more deeply the *average case* tractability of our approach (for which *worst-case* intractability is less relevant).

We are currently exploring methods of taking into account Tbox definitions $\mathcal{T}$ as background knowledge during refinement, as we plan to further reduce the non-minimality of our refinement operator w.r.t. $\mathcal{T} \neq \emptyset$ (a refinement step that is minimal w.r.t. $\mathcal{T} = \emptyset$ could be non-minimal when taking into account the definitions of some non-empty $\mathcal{T}$). Note that completeness is not an issue in this case, since a refinement operator that is complete w.r.t. $\mathcal{T} = \emptyset$ will remain complete w.r.t. a non-empty terminology.

Acknowledgments. The first author is grateful to Doina Ţilivea for discussions. Thanks are also due to the anonymous reviewers for their suggestions and constructive criticism.

References

1. Baader F., Küsters R. *Least common subsumer computation w.r.t. cyclic ALN-terminologies.* In Proc. Int. Workshop on Description Logics (DL'98), Trento, Italy.
2. Baader F., R. Küsters, R. Molitor. *Computing Least Common Subsumers in Description Logics with Existential Restrictions.* Proc. IJCAI'99, pp. 96-101.
3. Badea Liviu, Stanciu Monica. *Refinement Operators Can Be (Weakly) Perfect.* Proc. ILP-99, LNAI 1631, Springer, 1999, pp. 21-32.
4. Badea Liviu. *Perfect Refinement Operators Can Be Flexible.* Proc. ECAI-2000.
5. Borgida A. *On the relative Expressiveness of Description Logics and Predicate Logics.* Artificial Intelligence, Vol. 82, Number 1-2, pp. 353-367, 1996.
6. Buchheit M., F. Donini, A. Schaerf. *Decidable reasoning in terminological knowledge representation systems.* J. Artificial Intelligence Research, 1:109-138, 1993.
7. Cohen W.W., A. Borgida, H. Hirsh. *Computing least common subsumers in description logics.* Proc. AAAI-92, San Jose, California, 1992.
8. Cohen W.W., H. Hirsh. *Learning the CLASSIC description logic: Theoretical and experimental results.* In Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference, pp. 121-133, 1994.
9. Donini F.M., M. Lenzerini, D. Nardi, W. Nutt. *The complexity of concept languages.* Information and Computation, 134:1-58, 1997.
10. Donini F.M., M. Lenzerini, D. Nardi, A. Schaerf *AL-log: integrating datalog and description logics.* Journal of Intelligent Information Systems, 10:227-252, 1998.
11. Donini F.M., M. Lenzerini, D. Nardi, A. Schaerf, W. Nutt. *An epistemic operator for description logics.* Artificial Intelligence, 100 (1-2), 225-274, 1998.
12. Kietz J.U., Morik K. *A Polynomial Approach to the Constructive Induction of Structural Knowledge.* Machine Learning, Vol. 14, pp. 193-217, 1994.
13. Kietz J.U. *Some lower-bounds for the computational complexity of Inductive Logic Programming.* Proc. ECML'93, LNAI 667, Springer, 1993.
14. Levy A., M.C. Rousset. *CARIN: A Representation Language Combining Horn Rules and Description Logics.* Proc. ECAI-96, Budapest, 1996.
15. Levy A., M.C. Rousset. *The Limits on Combining Horn Rules with Description Logics.* Proc. AAAI-96, Portland, 1996.
16. Muggleton S. *Inverse entailment and Progol.* New Generation Computing Journal, 13:245-286, 1995.
17. van der Laag P., S.H. Nienhuys-Cheng. *A Note on Ideal Refinement Operators in Inductive Logic Programming.* Proceedings ILP-94, 247-260.
18. van der Laag P., S.H. Nienhuys-Cheng. *Existence and Nonexistence of Complete Refinement Operators.* ECML-94, 307-322.
19. Nienhuys-Cheng S.H., de Wolf R. *Foundations of Inductive Logic Programming.* LNAI 1228, Springer Verlag, 1997.
20. Nienhuys-Cheng S.-H., W. Van Laer, J. Ramon, and L. De Raedt. *Generalizing Refinement Operators to Learn Prenex Conjunctive Normal Forms.* Proc. ILP-99, LNAI 1631, Springer, 1999, pp. 245-256.

This article was processed using the LaTeX macro package with LLNCS style