Refining Concepts in Description Logics

Liviu Badea¹ and Shan-Hwei Nienhuys-Cheng²

¹ AI Lab, National Institute for Research & Development in Informatics
8-10 Averescu Blvd., Bucharest, Romania. e-mail: badea@ici.ro
² Erasmus University of Rotterdam
P.O. Box 1738, 3000 DR Rotterdam, the Netherlands. 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 studied to a lesser extent. 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 $\mathcal{ALC}$R description logic (to avoid overfitting, we disallow disjunctions from the target DL). Although no minimal refinement operators exist for $\mathcal{ALC}$R, 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 $\mathcal{ALC}$R 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 \([8]\) 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. 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 \([3, 4, 6]\). For example, Cohen and Hirsch’s LCSLEARN 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 \([9]\) and Progol, 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 \(\text{A}\text{L}\text{E}\text{R}\). This complete refinement operator is used by a more sophisticated learning algorithm to induce DL descriptions from examples.

## 2 The learning problem in Description Logics

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

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Syntax</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>top concept</td>
<td>(\top)</td>
<td>(\Delta)</td>
</tr>
<tr>
<td>bottom concept</td>
<td>(-)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>negation of atoms</td>
<td>(\neg A)</td>
<td>(\Delta \setminus A^\tau)</td>
</tr>
<tr>
<td>concept conjunction</td>
<td>(C_1 \sqcap C_2)</td>
<td>(C_1^\tau \cap C_2^\tau)</td>
</tr>
<tr>
<td>value restriction</td>
<td>(\forall R.C)</td>
<td>({x \in \Delta \mid \forall y,(x, y) \in R^\tau \rightarrow y \in C^\tau})</td>
</tr>
<tr>
<td>existential restriction</td>
<td>(\exists R.C)</td>
<td>({x \in \Delta \mid \exists y,(x, y) \in R^\tau \land y \in C^\tau})</td>
</tr>
<tr>
<td>role conjunction</td>
<td>(R_1 \sqcap R_2)</td>
<td>(R_1^\tau \cap R_2^\tau)</td>
</tr>
</tbody>
</table>

A knowledge base \(\mathcal{K} = (\mathcal{T}, \mathcal{A})\) has two components: a Tbox (terminology) \(\mathcal{T}\) containing a set of definitions (terminological axioms):

<table>
<thead>
<tr>
<th>Definition of (A)</th>
<th>Syntax</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sufficient</td>
<td>(A \leftarrow C)</td>
<td>(A^\tau \supseteq C^\tau)</td>
</tr>
<tr>
<td>necessary</td>
<td>(A \rightarrow C)</td>
<td>(A^\tau \subseteq C^\tau)</td>
</tr>
<tr>
<td>necessary and sufficient</td>
<td>(A = C)</td>
<td>(A^\tau = C^\tau)</td>
</tr>
</tbody>
</table>
and an Abox $\mathcal{A}$ containing extensional information in the form of membership assertions:

<table>
<thead>
<tr>
<th>Assertion instance</th>
<th>Syntax</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept instance $C(a)$</td>
<td>$a^I \in C^I$</td>
<td></td>
</tr>
<tr>
<td>role tuple $R(a, b)$</td>
<td>$(a^I, b^I) \in R^I$</td>
<td></td>
</tr>
</tbody>
</table>

The learning problem in DLs can be stated as follows.

**Definition 1** Let $K = \langle T, \mathcal{A} \rangle$ 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 $A$: $T'' = \{ A \leftarrow C_1, A \leftarrow C_2, \ldots \}$ that covers the examples $\mathcal{A}'$, i.e., $\langle T \cup T'', \mathcal{A} \setminus \mathcal{A}' \rangle \models \mathcal{A}'$.

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

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 An $\mathcal{ALER}$ refinement operator

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. 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 $\rho$ 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 $H \sim H'$ 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) $C_2$ ($C_1 \supseteq C_2$). 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 \( A \rightarrow C \). But unlike necessary definitions \( A \rightarrow C_1, \ldots, A \rightarrow C_n \), whose conjunction can be expressed as a single necessary definition \( A \rightarrow C_1 \cap \ldots \cap C_n \), 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: \( A \leftarrow C_1 \sqcup \ldots \sqcup C_n \).\(^1\)

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

**Definition 3** A downward refinement operator \( \rho \) 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 \( D \) (\( C \sqsupseteq D \)), 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\(^2\) 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\(^3\) and all its elements are incomparable.
- proper iff for all \( C \) and \( D \), \( D \in \rho(C) \) entails \( D \sqsubset C \) (or, equivalently, \( D \not\equiv C \)).

We first construct a complete but non-minimal (and thus highly redundant) 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.

The refinement operator \( \rho_0 \) 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 \( D \), \( C \sqsupseteq D \)):\(^4\)

\(^1\)Of course, such necessary definitions could be approximated by \( A \leftarrow \text{lcs}(C_1, \ldots, C_n) \), but this is more general than the conjunction of the original definitions.

\(^2\)A refinement chain from \( C \) to \( D \) is a sequence \( C_0, C_1, \ldots, C_n \) of hypotheses such that \( C = C_n, C_1 \in \rho(C_0), C_2 \in \rho(C_1), \ldots, C_n \in \rho(C_{n-1}) \), \( D = C_n \). Such a refinement chain does not "go through" \( E \) iff \( E \not\equiv C_i \) for \( i = 0, \ldots, n \).

\(^3\)\( D \) is a downward cover of \( C \) iff \( C \) is more general than \( D \) (\( C \sqsupseteq D \)) and no \( E \) satisfies \( C \sqsupseteq E \sqsupseteq D \).

\(^4\)We will modify this operator (while preserving its completeness) to reduce its non-minimality.
Refinement rules of $\rho_0$

- **[Lit]** $C \leadsto C \sqcap L$ with $L$ a DL-literal (to be defined below)
- **[3C]** $C \sqcap \exists R.C_1 \leadsto C \sqcap \exists R.C_2$ if $C_1 \leadsto C_2$
- **[3R]** $C \sqcap \exists R_1.D \leadsto C \sqcap \exists R_2.D$ if $R_1 \leadsto R_2$
- **[3E]** $C \sqcap \exists R_1.C_1 \sqcap \exists R_2.C_2 \leadsto C \sqcap \exists (R_1 \sqcap R_2).(C_1 \sqcap C_2)$
- **[∀C]** $C \sqcap \forall R.C_1 \leadsto C \sqcap \forall R.C_2$ if $C_1 \leadsto C_2$
- **[∀R]** $C \sqcap \forall R_1.D \leadsto C \sqcap \forall R_2.D$ if $R_2 \leadsto R_1$
- **[PR]** $R \leadsto R \sqcap P$ 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 left-to-right until they are no longer applicable:

- **[∀]** $\forall R.C \sqcap \forall R.D = \forall R.(C \sqcap D)$
- **[∀R]** $\forall R.C \sqcap \forall Q.D = \forall R.(C \sqcap D) \sqcap \forall Q.D$ if $R \sqsubseteq Q$
- **[∃]** $\exists R.C \sqcap \exists Q.D = \exists R.(C \sqcap D) \sqcap \exists Q.D$ if $R \sqsubseteq Q$

$\forall R.T = T \quad \exists R, - = -
C \sqcap \neg C = - \quad C \sqcap T = C \quad C \sqcap - = -$

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 $\forall (P_1 \sqcap P_2).\neg \forall P_1.(\neg A_1 \sqcap A_2) \sqcap \exists P_1.(\neg A_1 \sqcap A_2 \sqcap A_3)$.

**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 $\exists P.T$ for a primitive role $P$, or
- a value restriction $\forall \prod_{i=1}^{n} P_i.L'$ with $L'$ a DL-literal, where $\{P_1, \ldots, P_n\}$ is the set of all primitive roles occurring in the knowledge base.

An example of a $\rho_0$-chain: $\top \leadsto [\text{Lit}] A_1 \leadsto [\text{Lit}] A_1 \sqcap \forall (P_1 \sqcap P_2).A_2 \leadsto [\forall] A_1 \sqcap \forall (P_1 \sqcap P_2).A_2 \sqcap \exists P_2.A_3 \leadsto [\text{Lit}] A_1 \sqcap \forall (P_1 \sqcap P_2).A_2 \sqcap \exists P_2.A_3 \sqcap \exists P_2.A_4 \leadsto [\text{Lit}] A_1 \sqcap \forall (P_1 \sqcap P_2).A_2 \sqcap \exists P_2.A_3 \sqcap \exists P_2.A_4 \sqcap \exists P_2.(A_3 \sqcap A_4)$.

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 $\exists P.T$ for a primitive role $P$. But if $P$ already occurs in some other existential restriction on the

---

4 Under associativity, commutativity and idempotence of $\sqcap$. 

"top level" of $C$ (i.e. $C = C' \cap \exists R_1.C_1$ such that $P \supseteq R_1$, or, in other words, $R_1 = R'_1 \cap P$), then $\exists P.T$ is redundant w.r.t. $C$. (More generally, 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. $\exists R_1.C_1 \cap \exists R_2.C_2 \equiv \exists R_1.C_1$ if $C_1 \subseteq C_2$ and $R_1 \subseteq R_2$.)

But disallowing the addition of $\exists P.T$ to $C$ in cases in which some $\exists R_1.C_1$ with $R_1 \subseteq P$ already occurs in $C$ (which would ensure the properness of $\rho_0$) would unfortunately also lead to the incompleteness of $\rho_0$. For example, it would be impossible to reach $\exists P.A_1 \cap \exists P.A_2$ as a refinement of $\exists P.A_1$, because the first step in the following chain of refinements:

$$\exists P.A_1 \overset{[\text{Lit}]}{\leadsto} \exists P.A_1 \cap \exists P.T \overset{[\exists C]}{\leadsto} \exists P.A_1 \cap \exists P.A_2$$

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

$\exists P.T$ 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 $\rho$ to be improper (i.e. produce refinements $D \in \rho(C)$ that are equivalent to $C^5$), but – in order to obtain proper refinements – we will successively apply $\rho$ until a strict refinement (i.e. some $D \subset C$) is produced. The resulting refinement operator $\rho^i$ (the “closure” of $\rho$) will be proper, by construction. More precisely:

**Definition 5** $D \in \rho^i(C)$ iff there exists a refinement chain of $\rho$:

$$\begin{align*}
C & \overset{\varphi}{\leadsto} C_1 \overset{\varphi}{\leadsto} C_2 \overset{\varphi}{\leadsto} \ldots \overset{\varphi}{\leadsto} C_n = D,
\end{align*}$$

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

In the above-mentioned example, we have the following refinement chain of $\rho_0$:

$$C = \exists P.A_1 \overset{[\text{Lit}]}{\leadsto} C_1 = \exists P.A_1 \cap \exists P.T \overset{[\exists C]}{\leadsto} C_2 = \exists P.A_1 \cap \exists P.A_2$$

for which $C \equiv C_1$ and $C \subset C_2$. Therefore, $C_2 \in \rho_0^i(C)$ is a one-step refinement of the “closure” $\rho_0^i$.

For determining the most general value restrictions, we note that: $\forall R_1.C_1 \supseteq \forall R_2.C_2$ if $R_1 \supseteq R_2$ and $C_1 \supseteq C_2$. Thus, the restriction to be added $\forall R_2.C_2$ is

---

5The 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):

- $[\forall \text{red}] \quad \forall R_1.C_1 \cap \forall R_2.C_2 = \forall R_1.C_1$ if $R_1 \supseteq R_2$ and $C_1 \supseteq C_2$
- $[\exists \text{red}] \quad \exists R_1.C_1 \cap \exists R_2.C_2 = \exists R_1.C_1$ if $R_1 \supseteq R_2$ and $C_1 \supseteq C_2$

For example, their use would disallow obtaining $\exists P.A_1 \cap \exists P.A_2$ from $\exists P.A_1$: $\exists P.A_1 \overset{[\text{Lit}]}{\leadsto} \exists P.A_1 \cap \exists P.T \overset{[\exists C]}{\leadsto} \exists P.A_1 \cap \exists P.A_2$ because $\exists P.A_1 \cap \exists P.T$ would be simplified by the $[\exists \text{red}]$ redundancy elimination rule to $\exists P.A_1$, thereby making the second step ([\exists C]) inapplicable.
non-redundant w.r.t. $\forall R_1.C_1$ (i.e. $\forall R_1.C_1 \sqcap \forall R_2.C_2 \neq \forall R_1.C_1$ iff $R_1 \nsubseteq R_2$ or $C_1 \nsubseteq C_2$.)

Formally, the most general value restrictions take the form $\forall R.T$. But unfortunately, such value restrictions are redundant due to the identity $\forall R.T = T$. 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 $\forall \bigwedge_{i=1}^{n} P_i.L'$, but unfortunately this is, in general, also redundant. As shown above, redundancy can be eliminated by considering the “closure” $\rho_0^c$ of $\rho_0$.

3.1 Reducing non-minimality

It is relatively easy to show that the refinement operator $\rho_0$ (as well as its closure $\rho_0^c$) is complete. However, it is non-minimal (and thereby highly redundant), due to the following $A\subseteq R$ relationships:

$$\exists R.C_1 \sqcap \exists R.C_2 \equiv \exists R.(C_1 \sqcap C_2), \quad (1)$$

$$\forall R.C_1 \sqcap \forall R.C_2 = \forall R.(C_1 \sqcap C_2). \quad (2)$$

• (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 $C_1 \sqcap L$: $C \sqcap \exists R.C_1 \xrightarrow{[Lit]} C \sqcap \exists R.(C_1 \sqcap L)$, because this single-step refinement could also be obtained with the following sequence of smaller steps (the $[\exists \top]$ step is defined below):

$$C \sqcap \exists R.C_1 \xrightarrow{[Lit]} C \sqcap \exists R.C_1 \sqcap \exists R.\top \xrightarrow{[\exists \top]} C \sqcap \exists R.C_1 \sqcap \exists R.L \xrightarrow{[33]} C \sqcap \exists R.(C_1 \sqcap L).$$

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 $[33]$ (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 $C_1 \sqcap L$:

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

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

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

(In the last step, we applied the simplification rule $[\forall \forall]$.)
Thirdly, the relationship \( \exists R_1.C \sqcap \exists R_2.C \sqsupseteq \exists(R_1 \sqcap R_2).C \) suggests that, for reasons of minimality, we should also drop the \([\exists R]\) altogether. The rule \([\exists R]\) of \(\rho_0\) is redundant since the single step \(C \sqcap \exists R_1.D \rightarrow C \sqcap \exists(R_1 \sqcap P).D\) can also be obtained in several steps, as follows:

\[
C \sqcap \exists R_1.D \overset{[\text{Li}]}{\rightarrow} C \sqcap \exists R_1.D \sqcap P \sqcap \top \overset{[\text{Li}]}{\rightarrow} \ldots \rightarrow C \sqcap \exists R_1.D \sqcap P \sqcap D \overset{[\exists C]}{\rightarrow} C \sqcap \exists(R_1 \sqcap P).D.
\]

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 \([\exists C]\).

Finally, since the \([\text{Lit}]\) step of \(\rho_0\) 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 \(C \sim -\) is a valid refinement step (because \(C \supseteq -\)), it is not only non-minimal, but also useless if it is applied on the “top level” of the concept to be refined. However, refining some subconcept (of the concept to be refined) to \(-\) makes sense, for example \(\forall R.C \rightarrow \forall R.- (\forall R.- \text{ being consistent!})\), although it is still non-minimal. Unfortunately, as we show below, all refinements \(C_1 \rightarrow C_2\) that introduce a new \(-\) 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 \(C \sim -\), if not on the “top level” of the concept to be refined (i.e. in the rule \([\text{Lit}]\)) or in \([\exists C]\) (where allowing \(C \sqcap \exists R_1.C \rightarrow C \sqcap \exists R_1.- = -\) would lead to an inconsistency), then at least in \([\forall C]\), which has to be modified as follows:

\[
[\forall C] \quad C \sqcap \forall R.C_1 \sim C \sqcap \forall R.C_2 \quad \text{where} \quad C_1 \sim C_2 \quad \text{or} \quad C_2 = -.
\]

However, besides this explicit introduction of \(-\) 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 \((C \sim D)\) and \(D\) is consistent.

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

**Refinement rules of \(\rho'\)**

- \([\exists T]\) \(C \sqcap \exists R.T \rightarrow C \sqcap \exists R.L\) with \(L\) a DL-literal
- \([\exists C]\) \(C \sqcap \exists R.C_1 \sim C \sqcap \exists R.C_2\) if \(C_1 \not\equiv C_2\)
- \([\exists \exists]\) \(C \sqcap \exists R_1.C_1 \sqcap \exists R_2.C_2 \sim C \sqcap \exists(R_1 \sqcap R_2).(C_1 \sqcap C_2)\)
- \([\forall C]\) \(C \sqcap \forall R.C_1 \sim C \sqcap \forall R.C_2\) if \(C_1 \not\equiv C_2\) or \(C_2 = -\)
- \([\forall R]\) \(C \sqcap \forall R_1.D \sim C \sqcap \forall R_2.D\) if \(R_2 \sim R_1\)
- \([PR]\) \(R \sim R \sqcap P\) with \(P\) a primitive role.
In the following, we shall write \( C \xrightarrow{\text{Rule}} D \) whenever \( D \) is obtained as a refinement of \( C \) using the refinement rule \textbf{Rule} (which can be \([\exists T], [\exists C], [\exists], [\forall C], [\forall R],\) or \( - \) in the case of \( \rho \) – also \([\text{Lit}]\)). Moreover, we sometimes write \( C \xrightarrow{\sim} D \) instead of \( D \in \rho(C) \) (denoting the fact that \( D \) is obtained as a refinement of \( C \) without using the \([\text{Lit}]\) rule).

The refinement rules of the complete refinement operator \( \rho \) are the refinement rules of \( \rho' \) together with the \([\text{Lit}]\) rule.

\begin{itemize}
  \item **Refinement rules of \( \rho \)**
  \begin{align*}
    [\rho'] & \text{ refinement rules of } \rho' \\
    [\text{Lit}] & C \rightarrow C \cap L \text{ with } L \text{ a DL-literal such that } C \cap L \text{ is consistent}.
  \end{align*}
\end{itemize}

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

### 3.2 Properties of \( \rho \)

1. **Completeness.** Since \( \rho_0 \) is complete and the modification of \( \rho_0 \) to \( \rho \) preserves completeness, \( \rho \) will be complete too.

   Note that we are adding DL-literals either on the “top level” of the concept to be refined (using \([\text{Lit}]\)), or inside \( \exists R.T \) restrictions (in \([\exists T]\)). Such literals can be “moved inside” \( \forall \cdot \exists \cdot \forall \cdot \ldots \) chains by using the \([\exists] \) refinement rule and the \([\forall \forall] \) rewrite rule.

2. **Properness.** Like in the case of \( \rho_0 \), \( \rho \) is not proper because the DL-literals of the form \( \exists P.T \) inserted can be redundant. For achieving properness, we should consider the closure \( \rho^c \), rather than simplifying these redundancies (which would affect completeness).

3. **Minimality.** By construction, \( \rho \) has less non-minimal steps than \( \rho_0 \). However, there exists a fundamental trade-off between minimality and completeness of ALE\( R \) refinement operators in general (not just for ours).

**Proposition 6** There exist no minimal and complete ALE\( R \) refinement operators.

**Proof** There cannot exist a minimal refinement step \( C \rightarrow \forall P.\sim \), since there exists a sufficiently large \( n \) (for example, larger than the size of \( C \)) such that \( C \sqsupset C \cap \forall P_1 \ldots \forall P_n A \sqsupset \forall P \sim \).

However, we conjecture that although there exist no minimal and complete refinement operators for ALE\( R \), there exist refinement operators all
of whose steps $C \sim D$ are minimal, except for the steps involving the introduction of $-$ in some subconcept of $D$ (like in the $C_2 = -$ case of the $[\forall C]$ rule of our refinement operator $\rho$). This suggests that our $\rho$ 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 $\mathcal{A}\mathcal{L}\mathcal{E}\mathcal{R}$ refinement operators.

**Example 7** $A_1$ admits the following infinite set of minimal direct refinements: \{ $A_1 \cap \forall P.A$, $A_1 \cap \forall P.P.A$, $A_1 \cap \forall P.P.P.A$, ... $\}$. 

Therefore, since $\rho$ 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 $H$.)

## 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).

- 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 

  $$A(X) \leftarrow \text{forall}(R(X,Y),C(Y)).$$

   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 $A \leftarrow \forall K R.C$, where $K$ is an epistemic operator as in [5]. 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\textsuperscript{6}, the OWA is a problem when testing that a definition, for example $A \leftarrow \forall R.C$, covers a given

\textsuperscript{6} For example, when expressing constraints on role fillers using value restrictions.
example, for instance $A(a_i)$. Examples are unavoidably \textit{incomplete}. Even if all the \textit{known} $R$-fillers of $a_i$ from the Abox verify $C$:

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

this doesn’t mean that $a_i$ verifies $\forall R.C$, 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 $\forall K.R.C$ because all the \textit{known} $R$-fillers of $a_i$ verify $C$, so the definition $A \leftarrow \forall K.R.C$ 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 $K.R$, or, equivalently, assuming that the known fillers are all the fillers).

### 4.1 Example coverage for sufficient definitions

**Definition 8** In the case of a DL knowledge base $\langle \mathcal{T}, A \rangle$, for which $A' = \{ A(a_1), \ldots, A(a_p), \neg A(a_{p+1}), \ldots, \neg A(a_{p+n}) \} \subseteq A$ are considered (positive and negative) examples, we say that the sufficient definition $A \leftarrow C$ covers the (positive or negative) example $a_i$ iff

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

which is equivalent with the inconsistency of

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

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

**Example 9** Consider the following knowledge base $\langle \mathcal{T}, A \rangle$ with an empty Tbox ($\mathcal{T} = \emptyset$) and the Abox\footnote{because of the OWA, there could exist, in principle, a yet unknown $R$-filler of $a_i$ not verifying $C$.}

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

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

\footnote{Where the individuals $j, m, f, h$ stand for \textit{John, Mary, Fred, Helen}, while the atomic concepts \textit{IP, R, I} stand for \textit{Influential Person, Rich, Influential} and the primitive role \textit{F} for \textit{Friend}.}
The following sufficient definition covers all positive examples while avoiding all (not covering any) negative examples:

\[ IP \leftarrow R \cap \forall F.I \cap \exists F.I \]  

(3) covers the positive example \( IP(j) \) because \( \{ IP \leftarrow R \cap \forall F.I \cap \exists F.I \}, (A \setminus A') \cup \{ \neg IP(j) \} \) is inconsistent, since \( \neg R(j), (\neg \forall F.I)(j), (\neg \exists F.I)(j) \) are all inconsistent:

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

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

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

And since the more specific (3) does not cover the negative example \( \neg IP(f) \) (due to the consistency of \( (\neg \exists F.I)(f) \)), 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 (3).

The right-hand side of (3) is obtained by the refinement operator \( \rho \) from \( \top \) by the following sequence of steps:

\[
C_0 = \top \xrightarrow{[\text{Lit}]} C_1 = R \xrightarrow{[\text{Lit}]} C_2 = R \cap \forall F.I \xrightarrow{[\text{Lit}]} C_3 = R \cap \forall F.I \cap \exists F.I \xrightarrow{[\exists \text{v}]} R \cap \forall F.I \cap \exists F.I.
\]

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: \( \langle T, A \rangle \models (A \rightarrow C) \), which can be reduced to the inconsistency of \( A \cap \neg C \) w.r.t. \( \langle T, A \rangle \), i.e. to the non-existence of an instance
Such an \( x \) could be either \( a_i^3 \), 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, \( (A \land \neg C)(x) \) 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 \( A \land \neg C \)) only for the known examples \( a_i \) of \( A \). This amounts to proving \( (K A) \rightarrow C \), 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 \( \langle T, A \rangle \) iff \( \forall A(a_i) \in \mathcal{A}', cl(T, A \cup \{\neg C(a_i)\}) \) is inconsistent.

## 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 \( A \leftarrow T \), using a downward refinement operator 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.

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* \([4, 1]\)) of the Abox examples \( A(a_i) \). Most existing bottom-up approaches (such as \([4]\)) then use *least common subsumers* (LCS) \([3, 4, 2]\) 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 \( ALE \) 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

---

3 Assuming that the known positive examples of \( A \) are \( A' = \{A(a_1), \ldots, A(a_n)\} \).
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.

Since learning in even a very simple DL allowing for definitions of the form $C \leftarrow \exists R_c(A_1 \sqcap \ldots \sqcap A_n)$ is NP-hard (Theorem 2 of [7]), 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).

References


