# Measuring Similarity in Description Logics Using Refinement Operators*

Antonio A. Sánchez-Ruiz[1], Santiago Ontañón[2],
Pedro Antonio González-Calero[1], and Enric Plaza[2]

[1] Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid, Spain
antsanch@fdi.ucm.es, pedro@sip.ucm.es
[2] IIIA-CSIC, Artificial Intelligence Research Institute
Campus Univ. Aut. Barcelona, 08193 Bellaterra, Catalonia, Spain
{santi,enric}@iiia.csic.es

**Abstract.** Similarity assessment is a key operation in many artificial intelligence fields, such as case-based reasoning, instance-based learning, ontology matching, clustering, etc. This paper presents a novel measure for assessing similarity between individuals represented using Description Logic (DL). We will show how the ideas of *refinement operators* and *refinement graph*, originally introduced for inductive logic programming, can be used for assessing similarity in DL and also for abstracting away from the specific DL being used. Specifically, similarity of two individuals is assessed by first computing their *most specific concepts*, then the *least common subsumer* of these two concepts, and finally measuring their distances in the refinement graph.

## 1 Introduction

Description Logic (DL) [4] is becoming a de facto standard for knowledge representation in many application areas. DL constitutes a family of different logics, which have been carefully characterized in terms of expressivity and computational complexity of their deduction algorithms. Gaining momentum through the Semantic Web initiative, DL popularity is also related to a number of tools for knowledge acquisition and representation, as well as inference engines, that have been made publicly available. For these reasons, DL has also become the technology of choice for representing knowledge in knowledge-intensive case-based reasoning systems [23,9].

In the last few years, there has been a growing interest in defining similarity measures for expressive representation formalisms, such as DL. For example, Amato et al. [10] propose to measure concept similarity as a function of the intersection of their interpretations, which is, in fact, an approximation to the semantic similarity of concepts. The approximation is better or worse depending on how good is the sample of individuals used for assessing similarity. Thus, a good sample of individuals is required.

Other approaches have been proposed in order to assess similarity between individuals or concepts without requiring the use of a good sample of individuals. González

* Partially supported by the Spanish Ministry of Science and Education project Next-CBR (TIN2009-13692-C03-01 and TIN2009-13692-C03-03).

A. Ram and N. Wiratunga (Eds.): ICCBR 2011, LNAI 6880, pp. 289–303, 2011.

et al. [12] present a similarity measure for description logic designed for case-based reasoning systems. This similarity measure is based on the idea of hierarchical aggregation, in which the similarity between two instances is computed as an aggregation of the similarity of the values in their roles. Like most hierarchical aggregation measures, however, this measure has problems with roles which create cycles and they are ignored during similarity assessment. Related to the work of González et al. other similarity measures have been proposed using the hierarchical aggregation principle for other representation formalisms such as Horn Clauses [14], Feature Terms [1], or object-oriented representations [6]. Description logic has also been used to model CBR case retrieval mechanisms not based on similarity, but on the subsumption order [22].

The work presented in this paper is most related to that of Ontañón and Plaza [20], where they introduced two similarity measures for feature terms based on refinement graphs. The large differences between feature terms and description logic imply that their ideas cannot be applied directly, however. For instance, there are ideal refinement operators for feature terms, and also there is no distinction between concept and individual like in DL. In this paper we borrow the basic ideas and extend them in order to define similarity measures for description logic.

The rest of the paper runs as follows. The next section briefly introduces the basic concepts of DL and refinement operators used in the paper. Section 3 defines the proposed similarity measure, along with the algorithms used to compute it. Section 4 exemplifies the application of the similarity measure for a particular domain, while Section 5 presents the results of an empirical evaluation. Finally Section 6 concludes the paper and elaborates on future work.

## 2 Background

This section briefly summarizes basic concepts regarding description logic, refinement operators and similarity assessment, which we will use in this paper.

### 2.1 Description Logic

Description Logic (DL) is a family of knowledge representation languages which have received a lot of attention due to the development of the Semantic Web. DL is the logical foundation of OWL [13], the W3C standard ontology language, and consequently there is great interest in the creation and maintenance of knowledge bases coded using this formalism.

DL represents knowledge using three types of basic entities: *concepts*, *roles* and *individuals*. Concepts provide the domain vocabulary required to describe sets of individuals with common features, roles allow to describe relationships between individuals, and individuals represent concrete domain entities. DL expressions are built inductively starting from finite and disjoint sets of atomic concepts ($N_C$), atomic roles ($N_R$) and individual names ($N_I$).

An interpretation is a vector $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set called the *interpretation domain*, and $\cdot^{\mathcal{I}}$ is the interpretation function. The interpretation function relates each atomic concept $A \in N_C$ with a subset of $\Delta^{\mathcal{I}}$, each atomic role $R \in N_R$ with a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and each individual $a \in N_I$ with a single element of $\Delta^{\mathcal{I}}$.

**Table 1.** $\mathcal{EL}$ concepts and semantics

| Concept | Syntax | Semantics |
|---|---|---|
| Top concept | $\top$ | $\Delta^{\mathcal{I}}$ |
| Atomic concept | $A$ | $A^{\mathcal{I}}$ |
| Conjunction | $C \sqcap D$ | $C^{\mathcal{I}} \cap D^{\mathcal{I}}$ |
| Existential restriction | $\exists R.C$ | $\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$ |

**Table 2.** TBox axioms

| Axiom | Syntax | Semantics |
|---|---|---|
| Concept inclusion | $A \sqsubseteq B$ | $A^{\mathcal{I}} \subseteq B^{\mathcal{I}}$ |
| Disjointness | $A \sqcap B \equiv \bot$ | $A^{\mathcal{I}} \cap B^{\mathcal{I}} = \emptyset$ |
| Role domain | $domain(R) = A$ | $(x,y) \in R^{\mathcal{I}} \rightarrow x \in A^{\mathcal{I}}$ |
| Role range | $range(R) = A$ | $(x,y) \in R^{\mathcal{I}} \rightarrow y \in A^{\mathcal{I}}$ |

**Table 3.** ABox axioms

| Axiom | Syntax | Semantics |
|---|---|---|
| Concept instance | $C(a)$ | $a^{\mathcal{I}} \in C^{\mathcal{I}}$ |
| Role assertion | $R(a,b)$ | $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ |
| Same individual | $a = b$ | $a^{\mathcal{I}} = b^{\mathcal{I}}$ |
| Different individual | $a \neq b$ | $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ |

There are many DL with different expressive power and reasoning complexity depending on which concept constructs are allowed in the language[1]. In this paper we will focus on the $\mathcal{EL}$ logic, a light-weight DL with polynomial reasoning time that has proven to be useful to manage large knowledge bases in real world applications [8,2]. Table 1 shows the $\mathcal{EL}$ concept constructs as well as the extension of the interpretation function to complex concepts. Later in this paper we will use $\mathcal{C}(\mathcal{EL})$ to denote the set of all possible concept expressions that can be built in the $\mathcal{EL}$ logic.

A DL knowledge base (KB), $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, consists of two different types of information: the *TBox* or terminological component, which contains concept and role axioms and describes the domain vocabulary, and the *ABox* or assertional component, which uses the domain vocabulary to assert facts about individuals. For the purposes of this paper, a TBox is a finite set of concept and role axioms given in Table 2, and an ABox is a finite set of axioms about individuals shown in Table 3. We say that a *TBox* is *acyclic* if no concept definition depends directly or indirectly on itself. Note that we only allow concept inclusion axioms between atomic concepts and, therefore, these TBoxes will always be acyclic.

An interpretation $\mathcal{I}$ is a *model* of a knowledge base $\mathcal{K}$ iff the conditions described in Tables 2 and 3 are fulfilled for every axiom in $\mathcal{K}$. A concept $C$ is *satisfiable* w.r.t. a knowledge base $\mathcal{K}$ iff there is a model $\mathcal{I}$ of $\mathcal{K}$ such that $C^{\mathcal{I}} \neq \emptyset$.

The basic reasoning operation in DL is *subsumption*. Let $\mathcal{K}$ be a knowledge base and $C$ and $D$ be two concepts, we say that $C$ is subsumed by $D$ w.r.t. $\mathcal{K}$ ($C \sqsubseteq_{\mathcal{K}} D$)

[1] See `http://www.cs.man.ac.uk/~ezolin/dl/` for further information.

iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{K}$. When the knowledge base $\mathcal{K}$ is known we can simplify the notation and write $C \sqsubseteq D$. Finally, an *equivalence axiom* $C \equiv D$ is just an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$, and a *strict subsumption axiom* $C \sqsubset D$ is simply $C \sqsubseteq D$ and $C \not\equiv D$.

### 2.2 Refinement Operators

This section briefly summarizes the notion of *refinement operator* and introduces the concepts relevant for this paper (see [15] for a more in-depth analysis of refinement operators). Refinement operators are defined over *quasi-ordered sets*.

**Definition 1.** *A* quasi-ordered set *is a pair $(S, \leq)$, where $S$ is a set, and $\leq$ is a binary relation among elements of $S$ that is* reflexive *($a \leq a$) and* transitive *(if $a \leq b$ and $b \leq c$ then $a \leq c$).*

If $a \leq b$ and $b \leq a$, we say that $a \approx b$, or that they are *equivalent*.
Refinement operators are defined as follows:

**Definition 2.** *A* downward refinement operator $\rho$ *over a quasi-ordered set $(S, \leq)$ is a function such that $\forall a \in S : \rho(a) \subseteq \{b \in S | b \leq a\}$.*

**Definition 3.** *An* upward refinement operator $\gamma$ *over a quasi-ordered set $(S, \leq)$ is a function such that $\forall a \in S : \gamma(a) \subseteq \{b \in S | a \leq b\}$.*

In other words, upward refinement operators generate elements of $S$ which are "bigger" (which in this paper will mean "more general"), whereas downward refinement operators generate elements of $S$ which are "smaller" (which in this paper will mean "more specific"). Typically, the symbol $\gamma$ is used to symbolize upward refinement operators, and $\rho$ to symbolize either a downward refinement operator, or a refinement operator in general. A common use of refinement operators is for navigating sets in an orderly way, given a starting element. Typically, the following properties of operators are considered desirable:

- A refinement operator $\rho$ is *locally finite* if $\forall a \in S : \rho(a)$ is finite.
- A downward refinement operator $\rho$ is *complete* if $\forall a, b \in S | a \leq b : a \in \rho^*(b)$.
- An upward refinement operator $\gamma$ is *complete* if $\forall a, b \in S | a \leq b : b \in \gamma^*(a)$.
- A refinement operator $\rho$ is *proper* if $\forall a, b \in S\ b \in \rho(a) \Rightarrow a \not\approx b$.

where $\rho^*$ means the *transitive closure* of a refinement operator. Intuitively, *locally finiteness* means that the refinement operator is computable, *completeness* means we can generate, by refinement of $a$, any element of $S$ related to a given element $a$ by the order relation $\leq$ (except maybe those which are equivalent to $a$), and *properness* means that a refinement operator does not generate elements which are equivalent to a given element $a$. When a refinement operator is locally finite, complete and proper, we say that it is *ideal*. Other interesting properties of refinement operators have been discussed in the literature, such as *minimality* [5], but are not relevant for the purposes of this paper.

Concerning DL, the set of all the possible concept expressions and the subsumption relation between concepts form a quasi-ordered set and, therefore, we can define DL

$$\rho(C) = \rho_1(C) \cup \rho_2(C) \cup \rho_3(C) \cup \rho_4(C)$$

$$\rho_1(C) = \{C[A_i \rightarrow B] \mid B \in max\{B' \in N_C \mid B' \sqsubset A_i\}\}$$
$$\rho_2(C) = \{C \sqcap B \mid B \in max\{B' \in N_C \mid \forall A \in C : A \not\sqsubseteq B' \wedge B' \not\sqsubseteq A\}\}$$
$$\rho_3(C) = \{C[\exists R_i.D_j \rightarrow \exists R_i.E] \mid E \in \rho(D_j)\}$$
$$\rho_4(C) = \text{see Algorithm 1}$$

**Fig. 1.** Refinement operator

refinement operators to specialize or generalize concepts. In this paper we focus on the $\mathcal{EL}$ logic which has the advantage of having ideal refinement operators to traverse the quasi-ordered set $(\mathcal{C}(\mathcal{EL}), \sqsubseteq)$ [17].

It is also well known that there are no ideal refinement operators for the $\mathcal{ALC}$ logic, nor for any more expressive logic than that [18]. Fortunately, the similarity metric we describe in this paper does not require ideal operators and thus our approach is still valid for more expressive description logics.

### 2.3 A Refinement Operator for the $\mathcal{EL}$ Logic

Any $\mathcal{EL}$ concept $C$ can be written as a conjunction of concepts $C_1 \sqcap \ldots \sqcap C_n$ where each $C_i$ is either an atomic concept $A$ or an existential restriction $\exists R.D$ which filler $D$ follows the same rules. We say that $C_i$ is *redundant* in $C$ if there is another $C_j$ in $C$ such that $C_j \sqsubseteq C_i$ $(i \neq j)$, that is, if the information contained in $C_i$ is also in $C_j$. We say that a concept $C$ is *minimal* if it does not contain any redundant subconcept and the fillers of the existential restrictions are minimal as well. And, of course, any concept can be reduced to a minimal concept by removing the redundant information.

The refinement operator we propose, $\rho$, is shown in Figure 1 and it is proper only if it receives a minimal concept. We defined the operator as the union of four simpler operators ($\rho_1$, $\rho_2$, $\rho_3$ and $\rho_4$) that specialize the concept $C$ in different ways. The idea behind $\rho_1$ and $\rho_2$ is to specialize the original concept adding the most general atomic concepts which provide some new information. Symmetrically, $\rho_3$ and $\rho_4$ specialize the original concept adding the most general existential restrictions which provide some new information. Next, we describe each one of these operators in depth.

$\rho_1$ specializes a concept $C$ replacing any of its atomic concepts $A$ with one of its direct descendants in the conceptual hierarchy. For example, if there is a concept *Car* in the domain ontology, $\rho_1(Car)$ will return different types of car like *ShortCar* or *CloseCar*.

$\rho_2$ refines a concept $C$ adding the most general atomic concepts which neither subsume nor are subsumed by other atomic concept currently in $C$. For example, the operator $\rho_1(ShortCar)$ returns formulas like $ShortCar \sqcap CloseCar$. Both $\rho_1$ and $\rho_2$ can be easily computed by traversing the hierarchy of atomic concepts.

$\rho_3$ and $\rho_4$ follow the same ideas but they operate on existential restrictions rather than on atomic concepts. $\rho_3$ specializes existential restrictions currently in $C$ applying the refinement operator to their fillers. For example, if $C \equiv Car \sqcap \exists hasLoad.Load$

**Algorithm 1.** $\rho_4(C)$

```
1: RES = ∅
2: for R ∈ N_R do
3:    DS = {D | ∃R.D ∈ C}
4:    REM = {range(R)}
5:    while REM ≠ ∅ do
6:       E = pickOne(REM)
7:       if ∃D ∈ DS : D ⊏ E then
8:          REM = REM ∪ρ(E)
9:       else
10:         if ∄D ∈ DS : E ⊑ D then
11:            RES = RES ∪{C ⊓ ∃R.E}
12:         end if
13:      end if
14:   end while
15: end for
16: return RES
```

and the domain ontology describes different types of loads like *Triangle* or *Circle*, then $\rho_3(C)$ returns concepts like $Car \sqcap \exists hasLoad.Triangle$ and $Car \sqcap \exists hasLoad.Circle$.

Finally, $\rho_4$ refines a concept $C$ adding the most general existential restrictions which neither subsume nor are subsumed by other existential restrictions currently in $C$. Algorithm 1 shows how to compute these refinements. The idea is to find the most general fillers for each role which contribute some new information. The algorithm begins with the most general filler for each role, its range, and stores it in the set REM which contains the candidates that have not been processed yet. In each *while* loop the algorithm processes one of these remaining elements, $E$, according to the following ideas:

- if $E$ subsumes (is more general than) some of the existing fillers in $C$ then $E$ does not provide any new information yet and we need to keep specializing it, so we add all its refinements to REM.
- if $E$ is subsumed by (is more specific than) some of the existing fillers in $C$ then we ignore it because this situation is already covered by $\rho_3$.
- if $E$ does not subsume any of the existing fillers in $C$ and none of them subsumes $E$ then we have found a new interesting existential restriction and we add the corresponding formula to the solution set.

For example, $\rho_4(Car)$ will return formulas with new existential restrictions like $Car \sqcap \exists load.Shape$ or $Car \sqcap \exists wheels.Number$.

## 3 Measuring Similarity Using Refinement Operators

In this section we present our *DL refinement* ($S_{DL\rho}$) similarity measure for individuals in description logic which is based on the following intuitions:

First, given two concepts $C$ and $D$ such that $C \sqsubseteq D$, it is possible to reach $C$ from $D$ by applying a complete downward refinement operator $\rho$ to $D$ a finite number of times, i.e. $C \in \rho^*(D)$.

Second, the number of times a refinement operator needs to be applied to reach $C$ from $D$ is an indication of how much more specific $C$ is than $D$. In other words, the length of the refinement chain to reach $C$ from $D$, which we will note by $\lambda(D \xrightarrow{\rho} C)$, is an indication of how much more information $C$ has that was not contained in $D$. It is also an indication of their similarity: the smaller the length, the higher their similarity. Additionally, $\lambda(\top \xrightarrow{\rho} C)$ measures the distance from the most general concept, $\top$, to $C$, which is a measure of the amount of information in $C$.

Third, given any two concepts, their *least common subsumer* (LCS) is the most specific concept which subsumes both. The LCS of two concepts contains all that is shared between two concepts, and the more they share the more similar they are.

Using the previous three ideas, we can now define similarity between two concepts $C$ and $D$ as:

$$S^C_{DL\rho}(C, D) = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3}$$

where

$$\lambda_1 = \lambda(\top \xrightarrow{\rho} LCS(C, D))$$
$$\lambda_2 = \lambda(LCS(C, D) \xrightarrow{\rho} C)$$
$$\lambda_3 = \lambda(LCS(C, D) \xrightarrow{\rho} D)$$

Thus, the similarity between two concepts $C$ and $D$ is assessed as the amount of information contained in their LCS (i.e. the amount of information they share) divided by the total amount of information in $C$ and $D$ (the common information plus the information specific to each one).

Additionally, the same method can be used to assess similarity between individuals by adding an additional idea: that of the *most specific concept*. The most specific concept (MSC) of an individual is the most specific concept we can create in a given DL which contains the given individual, i.e. the concept in the DL which better represents such individual. Given two individuals $a$ and $b$, their similarity can be assessed in the following way:

$$S_{DL\rho}(a, b) = S^C_{DL\rho}(msc(a), msc(b))$$

The remainder of this section elaborates these ideas.

### 3.1 Most Specific Concepts

Let us start by briefly describing the idea of most specific concept (MSC). The MSC of a given individual is the most specific concept we can create which contains a given individual. It is well known that depending on the set of constructs allowed in the DL, the MSC might exist or not, and different algorithms have been proposed, like Baader's

**Algorithm 2.** msc(a,$\mathcal{A}$)

1: MSC = $\top$
2: **for** $C(a) \in \mathcal{A}$ **do**
3: MSC = MSC $\sqcap C$
4: **end for**
5: **for** $R(a,b) \in \mathcal{A}$ **do**
6: MSC = MSC $\sqcap(\exists R.msc(b,\mathcal{A}))$
7: **end for**
8: **return** MSC

for the $\mathcal{EL}$ logic [3]. This section presents a simple algorithm to approximate the MSC in $\mathcal{EL}$ assuming non-cyclic TBoxes (i.e. non-cyclic concept definitions).

Specifically, a concept $C$ is said to be the MSC of an individual $a$ with respect to an ABox $\mathcal{A}$, $msc_{\mathcal{A}}(a) = C$, if $C(a)$ and for each concept $D$ such that $D(a)$, $C \sqsubseteq D$ holds.

In general, the MSC does not always exist for a given individual in $\mathcal{EL}$. To illustrate why does this happen, let us consider the following example. Let $\mathcal{A} = \{R(a,a)\}$ be an ABox, and $n \geq 0$. It is easy to see that $a$ is an instance of the following concepts:

$$C_n \equiv \underbrace{\exists R. \ldots \exists R}_{n\ times}.\top$$

Notice that $C_i$ is more specific than $C_j$ if $i > j$, and thus, in this case, there is no concept which can satisfy the definition of MSC. This problem arises whenever there is a cycle in the definition of an individual. For simplicity reasons, in the remainder of this paper we will assume that individuals contain no cycles in their definition.

In the $\mathcal{EL}$ logic and under the assumption of no cycles we have used Algorithm 2 to compute the MSC of an individual. For example, given the following ABox which describes a train with one car which contains a triangle and a square:

*Train(t1), hasCar(t1,c1), Car(c1), hasLoad(c1,l1), Triangle(l1), hasLoad(c1,l2), Square(l2)*

our algorithm computes the following MSC of $t1$ that coincides with what was expected:

$$Train \sqcap \exists hasCar.(Car \sqcap \exists hasLoad.Triangle \sqcap \exists hasLoad.Square)$$

Notice that our acyclicity assumption does not restrict the application of the similarity measure presented in this paper; in case cycles are present, we would only need a different way of computing the MSC, like the one presented in [3].

### 3.2 Least Common Subsumer

Once we have the MSC of the two individuals we want to compare, the next step is to obtain the most specific concept that subsumes both, that is, their *least common subsumer* (LCS) [21].

**Algorithm 3.** lcs($C_1, ..., C_n$)

```
1: LCS = ⊤
2: while true do
3:    N = {C ∈ ρ(LSC)|∀_{i=1...n} C_i ⊑ C}
4:    if N = ∅ then
5:       return LCS
6:    else
7:       LCS = any C ∈ N
8:    end if
9: end while
```

**Definition 4.** *The* Least common subsumer *(LCS) of a set of given concepts, $C_1, ..., C_n$ is another concept $C = LCS(C_1, ..., C_n)$ such that $\forall_{i=1...n} C_i \sqsubseteq C$, and for any other concept $C'$ such that $\forall_{i=1...n} C_i \sqsubseteq C'$, $C \sqsubseteq C'$ holds.*

Depending on the specific DL being used, computing the LCS is trivial or not. In general, it can be computed by means of a search process, such as the one presented in Algorithm 3. Algorithm 3 works as follows. Initially, the LCS is set to the most general concept, $\top$. Then, the set of refinements of $\top$ that are still more general than all the concepts $C_1, ..., C_n$ is computed and stored in the set $N$. If $N$ is empty, we know that there are no refinements of the current $LCS$ that are still more general than all of the concepts $C_1, ..., C_n$, and thus, we have already found the LCS. If $N$ is non-empty, we can just select any of the concepts in $N$, and keep refining. In case the refinement operator is not proper, then which $C \in N$ is selected has to be carefully performed for not entering into an infinite loop.

For example, given two concepts $C_1 = Train \sqcap \exists hasCar.(\exists hasLoad.Triangle)$, and $C_1 = Train \sqcap \exists hasCar.(\exists hasLoad.Square)$, and assuming that the most specific concept that is more general than $Square$ and $Triangle$ in our ABox is $Shape$, we can use the previous algorithm to conclude that $msc(C_1, C_2) = Train \sqcap \exists hasCar.(\exists hasLoad.Shape)$.

### 3.3 Measuring Distances in the Refinement Graph

The last piece we require for defining $S_{DL\rho}$ is a way to measure the distance in the refinement graph between two concepts $C$ and $D$, such that $D \sqsubseteq C$, i.e. $\lambda(C \xrightarrow{\rho} D)$. This can be done by measuring the number of refinements required to reach $D$ from $C$.

**Algorithm 4.** $\lambda(C \xrightarrow{\rho} D)$

```
1: if C ≡ D then
2:    return 0
3: else
4:    C' ∈ {E ∈ ρ(C)|D ⊑ E}
5:    return 1 + λ(C' →ρ D)
6: end if
```

Computing the minimum number of refinements required to reach $D$ from $C$ might be computationally too expensive, so in $S_{DL\rho}$ we just use an estimate computed using Algorithm 4. Algorithm 4 works as follows. If $C$ is already equivalent to $D$, then their distance in the refinement graph is 0, otherwise, the algorithm takes one refinement $C'$ of $C$, and recursively computes the distance from $C'$ to $D$, the distance from $C$ to $D$ is then just 1 plus the distance from $C'$ to $D$.

## 4 Exemplification

This section shows an example of the similarity measure $S_{DL\rho}$ using the domain about trains introduced by Michalsky [16]. The two specific trains we are going to compare are shown in Figure 2. Both of them have just one short car that transports some type of load. The differences are that the car of $train1$ has a closed top and transports a triangle, while the car of $train2$ has an open top and transports a triangle and a circle.

In order to compute the similarity between both trains, we need to compute first the most specific concepts (MSC) that represents them using Algorithm 2:

$$msc_1 \equiv Train \sqcap \exists hasCar.(ClosedCar \sqcap ShortCar \sqcap \exists load.Triangle \sqcap \exists wheels.Two)$$
$$msc_2 \equiv Train \sqcap \exists hasCar.(OpenCar \sqcap ShortCar \sqcap \exists load.Triangle \sqcap \exists load.Circle \sqcap \exists wheels.Two)$$

Next we compute the most specific concept that subsumes the previous ones, that is $LCS(msc_1, msc_2)$, using Algorithm 3. This algorithm produces the following sequence of refinements:

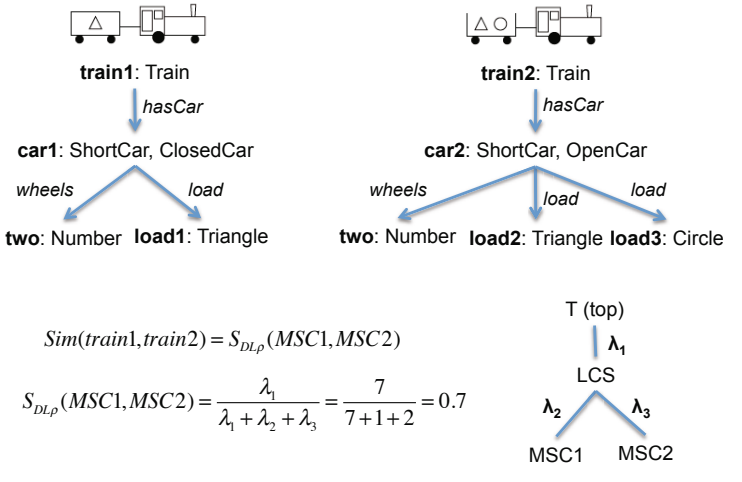


**Fig. 2.** Example of similarity between 2 trains

$0 : \top$
$1 : Train$
$2 : Train \sqcap \exists hasCar.Car$
$3 : Train \sqcap \exists hasCar.ShortCar$
$4 : Train \sqcap \exists hasCar.(ShortCar \sqcap \exists load.Shape)$
$5 : Train \sqcap \exists hasCar.(ShortCar \sqcap \exists load.Triangle)$
$6 : Train \sqcap \exists hasCar.(ShortCar \sqcap \exists load.Triangle \sqcap \exists wheels.Number)$
$7 : Train \sqcap \exists hasCar.(ShortCar \sqcap \exists load.Triangle \sqcap \exists wheels.Two)$

The LCS is the last concept in the previous sequence, and describes the information that is common to both trains: they have one short car with two wheels which transports a triangle. The amount of information shared by both trains can be measured as the length of the previous sequence ($\lambda_1 = 7$).

Then, we compute the amount of information that is specific to each train using Algorithm 4. First we search for a sequence of refinements from the LCS to $msc_1$ ($\lambda_2 = 1$):

$$1 : Train \sqcap \exists hasCar.(ClosedCar \sqcap ShortCar \sqcap \exists load.Triangle \sqcap \exists wheels.Two)$$

Next, we compute the sequence of refinements from the LCS to $msc_2$ ($\lambda_2 = 2$)):

$$1 : Train \sqcap \exists hasCar.(OpenCar \sqcap ShortCar \sqcap \exists load.Triangle \sqcap \exists wheels.Two)$$
$$2 : Train \sqcap \exists hasCar.(OpenCar \sqcap ShortCar \sqcap \exists load.Cricle \sqcap \exists load.Triangle \sqcap \exists wheels.Two)$$

Finally, the similarity between both trains is computed as follows:

$$S_{DL\rho}(train1, train2) = S^C_{DL\rho}(msc_1, msc_2) = \frac{\lambda_1}{\lambda_1 + \lambda_2 + \lambda_3} = \frac{7}{7+1+2} = 0.7$$

## 5 Empirical Evaluation

In order to evaluate our similarity measure, we used the trains data set shown in Figure 3 as presented by Michalski [16]. We selected this dataset since it is available in many representation formalisms (Horn clauses, feature terms and description logic), and therefore, we can compare our similarity measure with existing similarity measures in the literature. The dataset consists of 10 trains, 5 of them labelled as "west", and 5 of them labelled as "east."

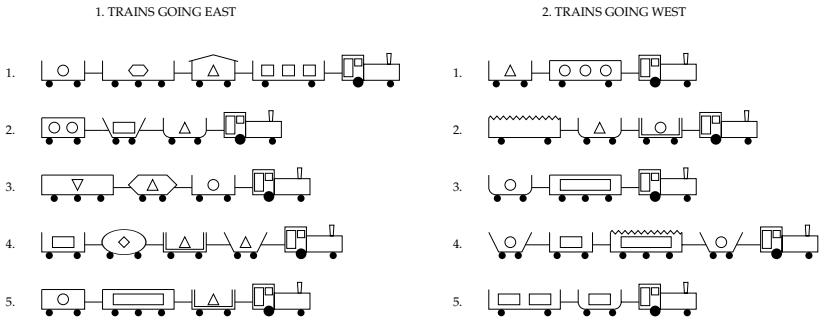


**Fig. 3.** Trains data set as introduced by Michalski [16]

We compared our similarity measure against 6 others: González et al. [12], a similarity measure for acyclic concepts in description logic, RIBL [11], which is a Horn clause similarity measure, SHAUD [1], which is a similarity measure for feature terms, and $S_\lambda$, $S_\pi$, and $S_{w\pi}$ [20], which are similarity measures for feature terms but also based on the idea of refinement operators. For RIBL, we used the original version of the trains dataset, for SHAUD, $S_\lambda$, $S_\pi$, and $S_{w\pi}$, we used the feature term version of the dataset used in [20], which is a direct conversion from the original Horn clause dataset without loss, and for our DL similarity measure (referred to as $S_{DL\rho}$ in Table 4), we used the version created by Lehmann and Hitzler [19].

We compared the similarity measures in 5 different ways:

- Classification accuracy of a nearest-neighbor algorithm.
- Classification accuracy of a 3-nearest neighbor algorithm.
- *Average best rank* of the first correct example: if we take one of the trains, and sort the rest of the trains according to their similarity with the selected train, which is the position in this list (rank) of the first train with the same solution as the selected train (west or east).
- Jaro-Winkler distance: the Jaro-Winkler measure [24] can be used to compare two orderings. We measure the similarity of the rankings generated by our similarity measure with the rankings generated with the other similarity measures.
- Mean-Square Difference (MSD): the mean square difference with respect to our similarity measure, $S_{DL\rho}$.

Table 4 shows the results we obtained by using a leave-one-out evaluation. Concerning classification accuracy, we can see that our similarity measure (labeled $S_{DL\rho}$ in the table) achieves a high classification accuracy, higher than most other similarity measures, except $S_{w\pi}$. The trains data-set is only apparently simple, since the classification criteria is a complex pattern which involves several elements from different cars in a train. The only similarity measure that came close is $S_{w\pi}$, which achieved an 80% accuracy (it misclassified trains west 1 and west 3). Concerning the average best rank, either the first or the second retrieved case using our similarity measure was always of the correct solution class, and thus it is very low, 1.4. Using the Jaro-Winkler similarity, and the

**Table 4.** Comparison of several similarity metrics in the trains dataset

| | $S_{DL\rho}$ | González et al. | RIBL | SHAUD | $S_\lambda$ | $S_\pi$ | $S_{w\pi}$ |
|---|---|---|---|---|---|---|---|
| Accuracy 1-NN | 70% | 50% | 60% | 50% | 40% | 50% | 80% |
| Accuracy 3-NN | 70% | 60% | 70% | 80% | 70% | 80% | 80% |
| Best Rank | 1.4 | 1.5 | 2.0 | 2.0 | 2.3 | 2.1 | 1.7 |
| Jaro-Winkler | - | 0.78 | 0.72 | 0.76 | 0.71 | 0.77 | 0.72 |
| MSD | - | 0.11 | 0.03 | 0.05 | 0.02 | 0.05 | 0.17 |

MSD, we can see that $S_{DL\rho}$ generates an order very similar to González et al.'s similarity, but that in terms of MSD, it is closest to $S_\lambda$, which is also based on refinement operators (although for feature terms instead of description logic).

Although a more thorough evaluation of our measure by integrating it into a real CBR system in a more complex task is part of our future work, our empirical evaluation shows promising results and confirms that refinement operators are a viable approach to assess similarity in CBR systems which use description logic as their representation formalism.

## 6 Conclusions and Future Work

We have presented the similarity measure $S_{DL\rho}$ for the $\mathcal{EL}$ description logic, based on notions of refinement graph and generalization space. Refinement graphs were introduced in a subset of Horn logic for the purpose of modeling inductive learning, until some of the authors of this paper [20] proposed they could be used for the purpose of estimating similarity in knowledge representation formalisms like description logic. Since that previous work presented $S_\lambda$, a similarity measure for feature terms, part of the claim was unsubstantiated until now, where $S_{DL\rho}$ is shown to be similarly defined for a given description logic.

Similarity is of great importance to CBR, and similarity for representation formalisms like description logic is important for knowledge-intensive CBR, but also for web-based applications, ontology alignment, and other tasks for AI systems. We consider this work a start into the process of achieving a better understanding of the relationship of case-based reasoning and the other fields of AI, like knowledge representation, logic, and inductive learning. More work need to be done, but this understanding might also be instrumental in greater visibility of CBR in the framework of artificial intelligence community.

Future work will focus on defining refinement-based similarity measures for more expressive description logics (DL) and also for subsets of Horn logics. Much of the work on the family of description logic revolves around finding subsets of DL that are expressive but computationally tractable; OWL, for instance, defines 3 language levels of increasing expressiveness and complexity. Defining refinement operators for high complexity DL may not be practical, so finding a more expressive subset of DL for which a tractable refinement-based similarity measure exists is our next goal.

## References


1. Armengol, E., Plaza, E.: Relational case-based reasoning for carcinogenic activity prediction. Artif. Intell. Rev. 20(1-2), 121–141 (2003)
2. Ashburner, M.: Gene ontology: Tool for the unification of biology. Nature Genetics 25, 25–29 (2000)
3. Baader, F.: Least common subsumers and most specific concepts in a description logic with existential restrictions and terminological cycles. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence, pp. 319–324. Morgan Kaufmann Publishers Inc., San Francisco (2003), http://portal.acm.org/citation.cfm?id=1630659.1630706
4. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, New York (2003)
5. Badea, L., Nienhuys-Cheng, S.H.: A refinement operator for description logics. In: Cussens, J., Frisch, A. (eds.) ILP 2000. LNCS (LNAI), vol. 1866, pp. 40–59. Springer, Heidelberg (2000)
6. Bergmann, R., Stahl, A.: Similarity measures for object-oriented case representations. In: Smyth, B., Cunningham, P. (eds.) EWCBR 1998. LNCS (LNAI), vol. 1488, pp. 8–13. Springer, Heidelberg (1998)
7. Blockeel, H., Ramon, J., Shavlik, J.W., Tadepalli, P.: ILP 2007. LNCS (LNAI), vol. 4894. Springer, Heidelberg (2008)
8. Bodenreider, O., Smith, B., Kumar, A., Burgun, A.: Investigating subsumption in SNOMED CT: An exploration into large description logic-based biomedical terminologies. Artif. Intell. Med. 39, 183–195 (2007), http://portal.acm.org/citation.cfm?id=1240342.1240604
9. Cojan, J., Lieber, J.: An algorithm for adapting cases represented in an expressive description logic. In: Bichindaritz, I., Montani, S. (eds.) ICCBR 2010. LNCS, vol. 6176, pp. 51–65. Springer, Heidelberg (2010)
10. d'Amato, C., Staab, S., Fanizzi, N.: On the influence of description logics ontologies on conceptual similarity. In: Gangemi, A., Euzenat, J. (eds.) EKAW 2008. LNCS (LNAI), vol. 5268, pp. 48–63. Springer, Heidelberg (2008)
11. Emde, W., Wettschereck, D.: Relational instance based learning. In: Saitta, L. (ed.) Machine Learning - Proceedings 13th International Conference on Machine Learning, pp. 122–130. Morgan Kaufmann Publishers, San Francisco (1996)
12. González-Calero, P.A., Díaz-Agudo, B., Gómez-Albarrán, M.: Applying DLs for retrieval in case-based reasoning. In: Proceedings of the 1999 Description Logics Workshop, DL 1999 (1999)
13. van Harmelen, F., McGuinness, D.L.: OWL Web Ontology Language Overview. W3C recommendation, W3C (February 2004), http://www.w3.org/TR/2004/REC-owl-features-20040210/
14. Horváth, T., Wrobel, S., Bohnebeck, U.: Relational instance-based learning with lists and terms. Machine Learning 43(1-2), 53–80 (2001)
15. van der Laag, P.R.J., Nienhuys-Cheng, S.H.: Completeness and properness of refinement operators in inductive logic programming. Journal of Logic Programming 34(3), 201–225 (1998)
16. Larson, J., Michalski, R.S.: Inductive inference of VL decision rules. SIGART. Bull. 63(63), 38–44 (1977)
17. Lehmann, J., Haase, C.: Ideal downward refinement in the EL description logic. In: Raedt, L.D. (ed.) ILP 2009. LNCS, vol. 5989, pp. 73–87. Springer, Heidelberg (2010)

18. Lehmann, J., Hitzler, P.: Foundations of refinement operators for description logics. In: Blockeel, H., et al. (eds.) [7], pp. 161–174
19. Lehmann, J., Hitzler, P.: A refinement operator based learning algorithm for the LC description logic. In: Blockeel, H., et al. (eds.) [7], pp. 147–160
20. Ontañón, S., Plaza, E.: On similarity measures based on a refinement lattice. In: Wilson, D., McGinty, L. (eds.) ICCBR 2009. LNCS, vol. 5650, pp. 240–255. Springer, Heidelberg (2009)
21. Plotkin, G.D.: A note on inductive generalization. In: Meltzer, B., Michie, D. (eds.) Machine Intelligence, vol. 5, pp. 153–163. Edinburgh University Press, Edinburgh (1970)
22. Salotti, S., Ventos, V.: Study and formalization of a case-based reasoning system using a description logic. In: Smyth, B., Cunningham, P. (eds.) EWCBR 1998. LNCS (LNAI), vol. 1488, pp. 286–297. Springer, Heidelberg (1998)
23. Sánchez-Ruiz-Granados, A.A., González-Calero, P.A., Díaz-Agudo, B.: Abstraction in knowledge-rich models for case-based planning. In: McGinty, L., Wilson, D.C. (eds.) ICCBR 2009. LNCS, vol. 5650, pp. 313–327. Springer, Heidelberg (2009)
24. Winkler, W.E., Thibaudeau, Y.: An application of the Fellegi-Sunter model of record linkage to the 1990 U.S. decennial census. In: U.S. Decennial Census. Technical report, US Bureau of the Census (1987)