The Knowledge Engineering Review, Vol. 36, e8, 1 of 30.
doi:10.1017/S0269888921000059

Learning multiple concepts in description logic through three perspectives

RAPHAEL MELO[1], KATE REVOREDO[2] and ALINE PAES[3]

[1]*IBM Research, Brazil*
E-mail: raphaelt@br.ibm.com
[2]*Institute for Data, Process and Knowledge Management, Vienna University of Economics and Business (WU), Vienna, Austria*
E-mail: kate.revoredo@wu.ac.at
[3]*Institute of Computing, Universidade Federal Fluminense (UFF), Niteroi, RJ, Brazil*
E-mail: alinepaes@ic.uff.br

Abstract

An ontology formalises a number of dependent and related concepts in a domain, encapsulated as a terminology. Manually defining such terminologies is a complex, time-consuming and error-prone task. Thus, there is great interest for strategies to learn terminologies automatically. However, most of the existing approaches induce a single concept definition at a time, disregarding dependencies that may exist among the concepts. As a consequence, terminologies that are difficult to interpret may be induced. Thus, systems capable of learning all concepts within a single task, respecting their dependency, are essential for reaching concise and readable ontologies. In this paper, we tackle this issue presenting three terminology learning strategies that aim at finding dependencies among concepts, before, during or after they have been defined. Experimental results show the advantages of regarding the dependencies among the concepts to achieve readable and concise terminologies, compared to a system that learns a single concept at a time. Moreover, the three strategies are compared and analysed towards discussing the strong and weak points of each one.

1 Introduction

Knowledge representation (Brachman & Levesque, 2004) is an essential component of applications that require reasoning over a domain. Description logics (DLs) (Baader & Nutt, 2010) form a family of representation languages, with different computational complexity and expressiveness, that are typically decidable fragments of first-order logic (FOL). With DL, it is possible to represent domain concepts and relations between them. Moreover, due to their computational and expressive power, they have been widely used to represent knowledge in the Semantic Web (Berners-Lee *et al*., 2001). Accordingly, DL provides the main underpinnings of the Ontology Web Language (OWL) (Antoniou & Van Harmelen, 2004; Hitzler *et al*., 2009), the language recommendation by W3C for describing ontologies in the Semantic Web.

One challenge imposed on the Semantic Web's realisation is the lack of widely available structured knowledge, primarily described in DL languages. One could then manually define the concepts and relations of a domain as a DL terminology. However, this process is time-consuming and error-prone (Maedche & Staab, 2001). Therefore, the consideration of mechanisms that have the ability of automatically learning a model represented in DL is relevant and sometimes even demanding.

Thus, a number of approaches to learn ontologies written in DL have been proposed in the last years (Iannone *et al*., 2007; Fanizzi *et al*., 2008; Lehmann & Hitzler, 2010; Bühmann *et al*., 2016; Konev

et al., 2017a; Funk *et al.*, 2019; Ozaki, 2020). However, the majority of them follow a single concept learning strategy, that is, they focus on learning piecewise concepts. Still, usually, a domain encompasses more than one non-primitive concept. In these cases, the primary way of using the existing tools is by learning each concept separately, ignoring the possible relations among those concepts. Consequently, the resulting terminology may be hard to visualise due to the presence of redundancy among the concepts' definitions, making the ontology larger than it could be. Consider, for example, the concepts of *mother* and *grandparent*. Arguably, saying that a grandparent is the parent of someone's mother is more direct than redefining the concepts of mother and father within the definition of *grandparent*.

In this paper, we argue that by learning the several concepts belonging to a single domain jointly, the resulting DL ontology is going to more reliable, concise, and readable, and, consequently, more useful to the user. Moreover, learning the concepts jointly may yield more accurate concepts than the standard piecewise approaches since each concept may have, in its definition, other concepts previously defined as a starting point.

Specifically, in this paper, we address the following research questions:

Q1: Does learning a terminology considering the concepts jointly make it have the same quality as learning a terminology considering the concepts independently?
Q2: Is a terminology that considers the relations among concepts more readable and concise than the terminology which neglect concepts dependency?
Q3: Does the quality improve when considering the dependencies among concepts?

To answer these questions, this manuscript presents three alternative approaches for handling multiple concepts learning in DL.

The first one, named *Concept Dependency through Examples (CDE)*, automatically establishes a suitable learning order by finding dependencies among the concepts from the examples. Next, this order is used to define which concept is going to be learned before the other, and so on. This approach was first discussed in Melo *et al.* (2013a), named there as *Before Terminology Learning*. Here, the presentation is further refined, and a sound evaluation methodology is used to answer research questions **Q1** and **Q2**.

The first phase of the CDE method aims at finding a taxonomy of the concepts, represented as a tree. However, it may exist some relation among the concepts that cannot be represented as a tree. For example, it may be the case that a concept is related to more than one concept and, therefore, alternative definitions could exist. As CDE provides a fixed ordering to guide the learning phase, before the learning process starts, the different relations will not be explored by the search algorithms.

Thus, the second approach for multiple concept learning (MCL), named *Concept Dependency through Assertions (CDA)*, is proposed. CDA permits learn a definition of a concept considering concepts whose definitions are still unknown. To accomplish that, the concepts that have not been learned yet are represented in the form of assertions and used as surrogates for the definitions of the concepts being learned. The use of such assertions allows for richer exploration in the search space and hence more diverse dependencies found between concepts than the CDE approach, as such dependencies are discovered *along* the learning phase, instead of beforehand. The idea for this approach was briefly presented as a work in progress in Melo *et al.* (2013b). Here, the method is detailed and novel experiments are performed, including a comparison with baseline approaches, to address research questions **Q1** and **Q2**.

The third method that we present in this work, named *Concept Dependency through Terminology (CDT)*, automatically rewrites the terminology by identifying dependencies according to possible redundancies within it. A redundancy is characterised when a section within a concept definition is equivalent to the full definition of another concept. The substitution of a section by its equivalent concept yields a more concise and equivalent terminology. This method is applied on an existing terminology, either it had been manually obtained or automatically learned. This method was first presented in Melo *et al.* (2014), where it was called Redundancy Removal on DLs Descriptions (R2D2). Here, we improve the formalism of the approach and the evaluation methodology to address the research question **Q2**.

Moreover, in this paper, all three approaches are compared, addressing research question **Q3**.
To sum up, in this manuscript, we contributed with:

(1) the presentation of a novel method that can learn multiple related concepts without pre-defining the concepts ordering (CDA);
(2) novel experiments for a complete evaluation of the previously presented method that automatically finds a pre-ordering of the concepts, and then learns a terminology with multiple related concepts using such an order (CDE);
(3) novel experiments for evaluating the method that compacts a previously defined terminology (CDT) and;
(4) a thorough comparison among the three approaches for learning a terminology regarding the concepts' dependency (CDA, CDE and CDT).

The paper is organised as follows. In Section 2, DL and concept learning are reviewed. In Section 3, our three approaches for MCL are described. Section 4 presents the experiment conducted to validate our proposals, together with a comparison among the three of them. Section 5 presents the works related to this research. Section 6 concludes the paper.

2 Background knowledge

In this section, we recall basic definitions of DL, followed by the presentation of formal definitions on the concept and terminology learning problems.

2.1 Description logics

DLs (Baader & Nutt, 2010) is the name given for a family of knowledge base representation (KR) formalisms used to represent domain knowledge by defining a terminology (the relevant concepts in the domain), then using these concepts to describe properties of individuals in the domain. The terminology and the information about the individuals form the knowledge base.

DLs are particularly useful for the Semantic Web because most of them are decidable fragments of the FOL. Hence, when the DL is decidable, a query is always answered, either positively or negatively. However, the *decidability* and *complexity* depend on the DL language chosen. Usually, there is a trade-off between expressiveness and the reasoning complexity, as a higher expressive power may incur in a higher reasoning complexity.

In a DL language, there are three types of entities: atomic concepts, atomic roles and individual names. *Atomic concepts* and *roles* are the elementary descriptions in a DL language. Complex descriptions, or *descriptions* in short form, can be built from atomic concepts and roles using concept *constructors*. DLs languages are distinguished by the set of constructors they provide; therefore, the expressiveness of a DL is tied to its set of constructors. In the remainder of this work, we use the letters a, b, a_1, b_1, ... for *individuals*; the letters A, B, A_1, B_1, ... for *atomic concepts*, the letters R, R_1, ... for *atomic roles* and the letters C, D, C_1, D_1, ... for *concept descriptions*. A *definition* has the form $A \equiv C$ or $A \sqsubseteq C$, although here we focus only on the first type. One can see that a definition has an atomic concept on its left side and a concept description on the right side. A *concept assertion* has the form $A(a)$ and a *role assertion* has the form $R(a, b)$.

In this manuscript, we adopted the $\mathcal{ALC}$(Attributive Concept Language with Complements) language (Baader & Nutt, 2010) motivated by the fact that it constitutes the foundation of other DLs. In this case, the syntax of the definitions belonging to the set of $\mathcal{ALC}$ concepts are as follows: (i) $\neg C$ is the negation of a concept; (ii) $C \sqcap D$ is the intersection of two concepts; (iii) $C \sqcup D$ is the union of two concepts; (iv) $\forall R.C$ is the universal restriction of a concept by a role; and (v) $\exists R.C$ is the existential restriction of a concept by a role.

The most frequently used way of defining $\mathcal{ALC}$ semantics relies on the definitions of interpretation, model and fixpoints over a domain (Baader & Nutt, 2010). Thus, here, the semantics of a description

Table 1 Example of a knowledge base comprising three components, a TBox (terminological axioms) and a ABox (assertional axioms), and RBox (relational axioms)

Knowledge base		
TBox	ABox	RBox
Father $\equiv$ Male $\sqcap \exists$ Parent.$\top$	Male(ALFRED)	Parent(BEATRICE, CORNELIA)
Mother $\equiv$ Female $\sqcap \exists$ Parent.$\top$	Male(CARL)	Parent(ALFRED, CORNELIA)
Wife $\equiv$ Female $\sqcap \exists$ Married.Male	Female(BEATRICE)	
Wife $\equiv$ Female $\sqcap \exists$ Married.Female	Female(CORNELIA)	
Husband $\equiv$ Male $\sqcap \exists$ Married.Female	Father(ALFRED)	
Husband $\equiv$ Male $\sqcap \exists$ Married.Male	Mother(CORNELIA)	

is given by a *domain* $\mathcal{DOM}$ (a set) and an *interpretation* $\cdot^{\mathcal{I}}$ (a functor). Individuals represent objects through names from a set $N_{\mathrm{I}} = \{a, b, \ldots\}$. Each *concept* in the set $N_{\mathrm{C}} = \{C, D, \ldots\}$ is interpreted as a subset of a domain $\mathcal{DOM}$. Each *role* in the set $N_{\mathrm{R}} = \{r, s, \ldots\}$ is interpreted as a binary relation on the domain.

An ontology in DL consists of a set of statements called as axioms (Krötzsch *et al.*, 2012). In DL, an ontology represents the knowledge base; therefore from now on, we use the expressions knowledge base and ontology interchangeably. Thus, a *knowledge base* in DL has three components: the *TBox*, the *ABox*, and the *RBox*. *TBox* axioms describe relationships (e.g. inclusion and equivalence) among concepts in the form of terminological knowledge, that is, the domain's vocabulary. *ABox* axioms accommodate assertions about named individuals using the terms defined in the terminology, aiming at expressing a relationship between a concept and an individual. *RBox* axioms assert relationships between named individuals. The Table 1 shows an example of a knowledge base with these three components.

In this paper, we assume the common assumptions made about DL terminologies: (i) there is only one definition for a concept name and (ii) the terminology is acyclic. Here, we say that a terminology is acyclic if there is no cycle in the taxonomy tree built from the concepts. Attached to a DL, there must be a reasoning mechanism, responsible for inferring knowledge about individuals using the knowledge base.

In this paper, we define the length of a description according do Definition 1.

DEFINITION 1 (Length of a description). *The length of a description C, denoted as Length(C), is the number of occurrence of constructors, atomic concepts, atomic roles and individuals after the equivalence/inclusion symbol.*

For example, considering the last concept definition exhibited in Table 1, *Husband* $\equiv$ *Male* $\sqcap \exists$ *Married.Male,* the *length(Husband)* $= 5$, which is the number of terms after the $\equiv$ symbol in the description *Husband*.

2.2 *Concept learning*

In order to effectively exploit terminologies over a domain, especially when dealing with large amounts of data such as in the Semantic Web (Berners-Lee *et al.*, 2001), it is useful to apply techniques from machine learning (Mitchell, 1997) for automatically inducing terminologies from data.

Since concepts are the basic component of a terminology, the first step is to conceive how to *automatically learn* a concept from available data. A number of algorithms together with implementations have been proposed in the literature to learn concepts in DL (Iannone *et al.*, 2007; Fanizzi *et al.*, 2008; Lehmann, 2009; Lehmann & Hitzler, 2010; Lima *et al.*, 2018; Ozaki, 2020). They are inspired on techniques developed within the field of *Inductive Learning Programming* (ILP) (De Raedt, 2008), whose general goal is to automatically induce logic programmes from data. When learning a concept, one has as goal to find a generalised and correct definition of such a concept from a set of examples, as defined below:

DEFINITION 2 **(Concept Learning in DL).**
Given:

- *a knowledge base* $\mathcal{KB} = \langle TBox, ABox, RBox \rangle$,
- *a concept named A whose definition is going to be learned,*
- *a set* $\mathcal{E}_\mathcal{A}$ *of concept assertions of the concept A, typically divided into positive assertions* ($\mathcal{E}_A^+ = \{A(a_1), \ldots, A(a_p)\}$) *and negative assertions* ($\mathcal{E}_A^- = \{\neg A(b_1), \ldots, \neg A(b_n)\}$), *such that* $\mathcal{E}_\mathcal{A} = \mathcal{E}_A^+ \cup \mathcal{E}_A^-$.

Find a definition $A \equiv C$ *such that:*

- $TBox \cup A \equiv C \models A(a) \forall A(a) \in \mathcal{E}_A^+$.
- $TBox \cup A \equiv C \not\models A(b) \forall A(b) \in \mathcal{E}_A^-$.

where C can be any possible description for A.

Although Definition 2 requires that the learned concept covers all the positive examples and none of the negative examples, usually this hard criteria is relaxed, to achieve a definition as good as possible.

Logic-based systems that learn concepts often compare candidates through scoring functions based on the number of positive/negative examples covered. In DL-Learner, for example, a fitness relationship considers the number of positive examples and negative examples covered as well as the length of solutions when expanding candidates in the tree search.

2.2.1 Terminology learning

DEFINITION 3 **(Terminology Learning).** *Consider a terminology learning task* $\mathcal{TL} = \langle \mathcal{KB}, \{LT_{A_1}, \ldots, LT_{A_m}\} \rangle$, *where* $LT_{A_i} \in \mathcal{TL}$ *encompasses a concept name and a set of examples such that* $LT_{A_i} = \langle A_i, \mathcal{E}_{A_i}^+, \mathcal{E}_{A_i}^- \rangle$ *and* $\mathcal{KB} = \langle TBox, ABox, RBox \rangle$.

Learning a terminology $\mathcal{T}$ *means for each* $LT_{A_i} \in \mathcal{TL}$ *find a concept definition* $A_i \equiv C_i$, *where:* $TBox \cup A_i \equiv \mathcal{C}_i \models A_i(a_j)\ \forall A_i(a_j) \in \mathcal{E}_{A_i}^+$, *and* $TBox \cup A_i \equiv \mathcal{C}_i \not\models A_i(b_j)\ \forall A_i(b_j) \in \mathcal{E}_{A_i}^-$, *such that* $\mathcal{T} = \{A_1 \equiv C_1, \ldots A_m \equiv C_m\}$.

When approaching the task of learning a terminology, one could use two general solutions: i) learning the concepts without including dependencies between them (independence assumption) or ii) learn the concepts regarding dependencies. The first one could be solved by defining a terminology learning task as a set of independent concept learning tasks. In this paper, we address the second case, henceforth called MCL. We argue that the interpretability of a terminology $\mathcal{T}$ is directly related to the dependencies among concepts found via the concept learning tasks. Moreover, the interpretability also benefits from the simplicity of the terminology.

We follow the same line of thought as Sommer (1995), which stated some intuitions on how simplifying a theory can cause it to be more readable, summarised as follows: (i) a deeper[1] theory is preferable to a shallow one, as in this way one can construct concepts over other concepts; (ii) the longer is the concept, the harder it is to understand it; and (iii) the more redundant a theory is, the less inclined one would be to accept it.

Thus, simplicity by itself is a worthy goal of machine learning; however, it should not be argued that choosing a simpler set of definitions over a more complex always yields a better accuracy (Domingos, 1998). In this work, we have the main goal of learning sets of definitions that are more readable (shorter) while still maintaining a comparable accuracy.

3 Multiple concept learning

In this section, we describe our three approaches for learning multiple DL concepts.

[1] The depth of a theory is the maximum number of inferences required to answer a goal query.

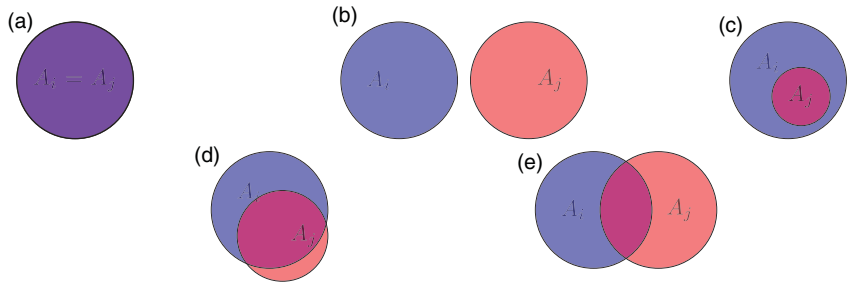


Figure 1 The relationship among concepts A_i and A_j through their positive examples sets, represented as Venn diagrams

3.1 Finding concept dependencies through examples (CDE)

The CDE method finds a suitable learning order, according to a dependency taxonomy built from the examples. In this way, a concept learned at iteration i can be included in the definition of concepts learned after iteration i. This method relies on the assumption that by finding out the subsumption relationship between concepts, it is possible to yield an ordering of the concepts, which in return will support the learning of a rich — although simple and easily readable — terminology.

The question that arises is how to obtain a subsumption order from concepts that do not have definitions yet, that is, *before* their definitions are learned. We tackle this problem by developing a procedure capable of finding out the relations among concepts *before* learning them, by taking advantage of their set of examples. Because of the completeness and consistency properties of the concept learning task, the set of positive and negative examples provides strong evidence for the existence of dependencies and relationships among concepts.

Let $LT_{Ai} =< \mathcal{KB}, \mathcal{E}^+_{A_i} \cup \mathcal{E}^-_{A_i} >$ and $LT_{Aj} =< \mathcal{KB}, \mathcal{E}^+_{A_j} \cup \mathcal{E}^-_{A_j} >$ concept learning tasks for concepts A_i and A_j, respectively. After learning descriptions for these concepts, their definitions will be part of the *TBox*. Therefore, a comparative analysis of the individuals belonging to $\mathcal{E}^+_{A_i}$ and $\mathcal{E}^+_{A_j}$ could point out how strongly related the corresponding concepts are and what might be the best order ($\preccurlyeq$) for learning them. Briefly, we argue that high similarity on the positive examples sets indicates high similarity between concept definitions. Example 1 illustrate this point.

Example 1. Let $LT_{\text{FATHER}} =< \mathcal{KB}, \mathcal{E}^+_{\text{FATHER}} \cup \mathcal{E}^-_{\text{FATHER}} >$ and $LT_{\text{GRANDFATHER}} =< \mathcal{KB}, \mathcal{E}^+_{\text{GRANDFATHER}} \cup \mathcal{E}^-_{\text{GRANDFATHER}} >$, where $\mathcal{E}^+_{\text{FATHER}} = \{\text{john,bob,edd}\}$ and $\mathcal{E}^+_{\text{GRANDFATHER}} = \{\text{bob}\}$. $\mathcal{E}^+_{\text{GRANDFATHER}} \subseteq \mathcal{E}^+_{\text{FATHER}}$, then the definition of concept FATHER includes more individuals than the definition of concept GRANDFATHER. Thus, the definition of FATHER is more general than the definition of GRANDFATHER. Therefore, the concept FATHER can be used within the description of GRANDFATHER, but not the other way around. Thus, if the concept FATHER is learned and added to $\mathcal{KB}$ before learning the concept GRANDFATHER, then FATHER can be used to compose the description of GRANDFATHER. A learning order is then established between the two concepts: FATHER $\preccurlyeq$ GRANDFATHER.

The comparative analysis between two concepts (A_i and A_j) through their sets of positive examples comprises the following five cases, illustrated at Figure 1:

(a) Identical sets ($A_i \equiv A_j$): all elements belonging to one set also belong to the other set;
(b) Disjoint sets ($A_i \cap A_j = \emptyset$): both sets have no elements in common;
(c) Inner set ($A_i \subseteq A_j$): one set is a subset of the other;
(d) High intersection: the proportion of shared elements overcomes a threshold;
(e) Low intersection: the proportion of shared elements does not overcome a threshold.

We claim that the higher is the intersection between sets of positive examples, the greater is the relationship between the concepts. Thus, case (a) describes an upper limit, where the concepts are totally

Algorithm 1 Algorithm for learning a taxonomy of concepts

Require: a terminology learning task ($\mathcal{TL} = \langle LT_{A_1}, \ldots, LT_{A_m} \rangle$) and a knowledge base $\mathcal{KB} = \langle TBox, ABox, RBox \rangle$, an integer *threshold*
Ensure: a taxonomy of concepts $\mathcal{TA}$

1: Starts $\mathcal{TA}$ with the concept $\top$ in the root;
2: **for** each $LT_{A_i} = \langle A_i, \mathcal{E}_{A_i} \rangle$ **do**
3: add A_i as a child of the root;
4: **for** each pair (LT_{A_i}, LT_{A_j}) of learning tasks in $\mathcal{TL}$ **do**
5: $\mathcal{TA} \leftarrow$ call Algorithm 2 (LT_{A_i}, LT_{A_j}, $\mathcal{TA}$, *threshold*)
6: **return** $\mathcal{TA}$

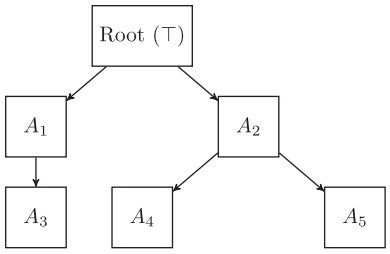


Figure 2 An example of a taxonomy depicting the relation among concepts A_1, A_2, A_3, A_4 and A_5

related, while case (b) indicates a lower limit, where the two concepts have no relationship. Furthermore, cases (a), (b) and (c) are the ideal cases because the underlying relationships between concepts are clear; on the other hand, the cases (d) and (e) need some mapping to one of the ideal cases, as 'high' and 'low' depends upon the observer. This is accomplished via a threshold value that will indicate how the concepts are related to each other. Therefore, the case (c) is a special case of (d), when the threshold is equal to 1; the case (b) is a special case of (e), when the threshold is equal to 0.

In this way, the relations among all concept definitions may be obtained from the comparative analysis of each pair of concepts that belong to the terminology learning task. The resulting explicit relations among the concepts are translated to a subsumption tree (*a taxonomy*). For instance, consider the taxonomy depicted in Figure 2 for the set of concepts $N_C = \{A_1, A_2, A_3, A_4, A_5\}$.

The root node indicates the set of all individuals of the domain ($\top$ concept). A concept in a branch on level i is more general than a concept on the same branch in level j ($i < j$), thus concept A_1 is more general than concept A_3. Assuming that a more general concept can be used in the definition of a more specialised one, the former should be learned first. Thus, the resulting taxonomy of the concepts can be used to determine the best ordering to learn the concept definitions.

3.1.1 Discovering a concept ordering

Algorithm 1 presents the top-level procedure for building a taxonomy of concepts that is used to define an order for learning the concepts of a terminology. It is based on the five possible cases previously discussed.

The algorithm receives as input a terminology learning task, as outlined in Definition 3, and returns a taxonomy of its concepts. Given the ordering tree, a concept can only be defined if all its parents have already been defined. The root of the tree is a default concept which must be the parent of all the concepts in $\mathcal{TL}$. This algorithm starts by creating a tree with only the root. Next, each concept (A_i) corresponding learning task (LT_{Ai}), where $LT_{Ai} \in \mathcal{TL}$, is included in the tree as child of the root. Then, Algorithm 2 is called for each pair of concepts A_i and A_j in order to find their relationship. After checking each possible dependency, the final taxonomy of concepts is returned.

Algorithm 2 Algorithm for ascertaining dependencies among concepts in a terminology

Require: Two concept learning tasks $LT_{A_i} = \langle A_i, \mathcal{E}^+_{A_i}, \mathcal{E}^-_{A_i} \rangle$ and $LT_{A_j} = \langle A_j, \mathcal{E}^+_{A_j}, \mathcal{E}^-_{A_j}$; a taxonomy $\mathcal{TA}$, a threshold Te.
Ensure: The updated taxonomy $\mathcal{TA}$

1: **if** $A_i \equiv A_j$ **then return** $\mathcal{TA}$
2: shared-individuals $\leftarrow |\mathcal{E}^+_{A_i} \cap \mathcal{E}^+_{A_j}|$
3: **if** shared-individuals $= 0$ **then return** $\mathcal{TA}$
4: **if** $|\mathcal{E}_{A_i}+| \leq |\mathcal{E}_{A_j}+|$ **then**
5: smaller_concept $\leftarrow A_i$; larger_concept $\leftarrow A_j$
6: **else**
7: smaller_concept $\leftarrow A_j$; larger_concept $\leftarrow A_i$
8: **if** (shared-individuals / |smaller_concept| $\geq Te$) **then**
9: include larger_concept in the parents set of smaller_concept in $\mathcal{TA}$
10: remove ROOT from the parents set of smaller_concept
11: **return** $\mathcal{TA}$

Algorithm 3 Top-level algorithm for learning terminologies

Require: A terminology learning task $\mathcal{TL}$.
Ensure: A Terminology $\mathcal{T}$

1: find a Taxonomic Tree $\mathcal{TA}$ through Algorithm 1
2: find an Ordered Sequence A_S of concept learning tasks $\mathcal{TL}$ using $\mathcal{TA}$, where concepts only come after their parents
3: **for** each $LTA_i \in A_S$, in the established order **do**
4: learn a description C_i using LT_{A_i} and $\mathcal{KB}$, using any algorithm that learns concepts in Description Logics
5: include $A_i \equiv C_i$ in *TBox*
6: **return** $\mathcal{T}$

In Algorithm 2, the relationship between the positive examples sets of the two concepts are analysed in order to resolve in which of the five cases discussed above it belongs. In case a pair of concepts are evaluated as equivalent, both of them are kept in the tree, making possible to learn a proper concept to each of them. It may be the case that during the learning, the negative examples point out that they have some difference. This also happens when the set of examples of two concepts is disjoint (Völker *et al.*, 2015).

When a set of examples is a subset of another one, we assume that the concept with the super set is more general than the other, called here as a *subsumer*. The concept with the subset is called *subsumee* and put as a child of the *subsumer* in the tree. Thus, the edge connecting the *subsumee* concept and the root is removed.

Finally, by employing a threshold parameter, we decide the type of relationship of two concepts that have intersected sets. If the proportion of shared individuals is higher than a threshold, we assume that the concept with more examples is a *subsumer* while the other is a *subsumee*.

3.1.2 Learning terminologies through concept ordering

Algorithm 3 presents the overall procedure for learning a terminology using the CDE method. It requires as input the terminology learning task $\mathcal{TL}$. The first step of the algorithm is to call the procedure devised as Algorithm 1, which is going to return the taxonomy tree comprising each concept addressed by the terminology problem.

Next, to find the order that the concepts are going to be learned, the nodes are collected in a breadth-first manner. For instance, considering the taxonomy in Figure 2, the concepts would be learned in the order $< A_1, A_2, A_3, A_4, A_5 >$. In this way, it is possible to guarantee that a concept is only handled after the descriptions of all its parents have been learned either.

Finally, to learn a description, the concept itself, its positive and negative examples and the current background knowledge are submitted to a DL learning algorithm, such as DL-Learner (Lehmann & Hitzler, 2010), DL-FOIL (Fanizzi *et al*., 2008) or Yin-Yang (Iannone *et al*., 2007). Then, the description found is included in the TBox of each $\mathcal{KB}$ of the following LT_{Ai}, so that it can be used in next iterations of the loop, as part of other descriptions.

3.2 Finding dependencies during the learning phase (CDA)

The CDE approach defines an arguably ideal dependency order for the concepts and strictly follows such an order when learning the descriptions. This is required by the learning phase as it is not possible to use an 'unknown' non-atomic concept, that is, a non-atomic concept without a description, when inducing a description for another concept. However, defining such an order before the learning phase has the disadvantage of not taking into account what is actually being learned to find possible dependencies between concepts. Thus, one question that arises is how the definition for a concept A_j can be considered in the description of another concept A_i before the description for A_j has been actually learned? The way we answer this question is devising the procedure *CDA*, that makes the concepts look like atomic ones. For achieving that, the examples of a concept are transformed into assertions in the *ABox*, defining a new *ABox*: $ABox' = ABox \cup \mathcal{E}^+_{A_i} \cup \neg\mathcal{E}^-_{A_i}$, where $\neg\mathcal{E}^-_{A_i}$ is the set of negative assertions for the concept A_i; it specifies the individuals that are not elements of A_i.

CDA top-level procedure (Algorithm 4) works like that: for a particular learning task $LT_{Ai} \in \mathcal{TL}$, all other concepts A_j, where $LT_{Aj} \in \mathcal{TL} \setminus LT_{Ai}$ and A_j was not already learned, are transformed into atomic concepts available to be used during the learning phase for A_i (Algorithm 5). For the successful use of this approach, there are some conditions that must be observed, as stated in Restriction 1.

RESTRICTION 1 (CDA's Restrictions) *The following are restrictions for the proper work of the method CDA:*

1. *All the concept learning tasks must share the same ABox, that is, they all should encompass the same set of individuals.*
2. *All the relevant individuals for a particular concept learning task should appear in one of its example sets (completeness).*
3. *The background knowledge is a finite model.*
4. *For every concept A in terminology learning task $\mathcal{TL}$ and every individual I relevant to the concept A, $I \in \mathcal{E}^+_A$ or $I \in \mathcal{E}^-_A$.*

Those are reasonable constraints as the amount of information on $\mathcal{E}$ tends to yield better results[2].

When CDA's restrictions are uphold, it is possible to use the sets of examples $\mathcal{E}_{A_i}$ of each concept learning task within a terminology learning task $\mathcal{TL} =< LT_{A1}, \ldots, LT_{Am} >$, as a surrogate for the definition that will be found later, with no semantic lost. There is no difference with regard to the result obtained from the reasoning services for a query $A(a)$, whether it is an assertion (ABox) or if it follows an inference process using the terminology (TBox). An additional benefit is that each concept learning task can be executed independently, allowing for parallel learning of each concept.

Moreover, CDA procedure can overcome CDE bias for concept/subconcept ordering which may yield more concise terminologies. CDE is tuned to find relations of the type *subsumer* and *subsumee*, for example, in the kinship domain we have GRANDFATHER $\sqsubseteq$ FATHER. However, it cannot reliably work with different relations, for example, the union of GRANDFATHER and GRANDMOTHER to

[2] This is a general principle on Machine Learning: The amount of relevant information positively affects the quality of learned models.

Algorithm 4 CDA (avoiding cycles)

Require: $\mathcal{TL} = < LT_{A_i}, \ldots, LT_{A_m} >$, where $LT_{A_i} = \langle A_i, \mathcal{E}^+_{A_i}, \mathcal{E}^-_{A_i} \rangle$; $\mathcal{KB} = \langle ABox, TBox, RBox \rangle$.
Ensure: A Terminology $\mathcal{T}$

1: $\mathcal{T} = TBox$; $R = \mathcal{TL}$
2: **for** i = 1 to n **do**
3: $listOfDependents_i = \emptyset$
4: **for** each $LT_{A_i} \in \mathcal{TL}$ **do**
5: $R \leftarrow R - LT_{A_i}$;
6: $ABox_{A_i} \leftarrow$ call Algorithm 5 (R, A_i, $ABox$, $listOfDependents_i$)
7: learn a description C_i for A_i, considering $\mathcal{KB}_{A_i} = \langle ABox_{A_i}, \mathcal{T}, RBox \rangle$, using any algorithm that learns piece-wise concepts in DL;
8: include $A_i \equiv C_i$ in $\mathcal{T}$;
9: **for** each $LT_{A_j} \in R$ **do**
10: **if** C_i uses A_j **then**
11: $listOfDependents_j \leftarrow listOfDependents_i \cup LT_{A_i}$
12: **return** $\mathcal{T}$

Algorithm 5 Algorithm to find virtual atomic concepts

Require: $R = \langle LT_1, \ldots, LT_m \rangle$, where $LT_i = \langle A_i, \mathcal{E}^+_{A_i}, \mathcal{E}^-_{A_i} \rangle$; A, $ABox$, $listOfDependents$.
Ensure: $ABox_f$

1: $ABox_f \leftarrow \emptyset$
2: **for** each assertion $A(a) \in ABox$ **do**
3: $ABox_f \leftarrow ABox_f \cup A(a)$
4: **for** each $LT_j \in R$ **do**
5: **if** $LT_j \notin listOfDependents$ **then**
6: $ABox_f \leftarrow ABox_f \cup \mathcal{E}^+_{A_j} \cup \neg\mathcal{E}^-_{A_j}$
7: **return** $ABox_f$

define GRANDPARENT. CDA is capable of dealing with it because we turn the learning task into the responsible component for finding relationships between concepts.

Example 2 shows the output of the terminology learning task for the set of concepts {GRANDPARENT(GP), GRANDMOTHER(GM), GRANDFATHER(GF)}.

Example 2. Let $\mathcal{KB}$ be as presented in Figure 1, $\mathcal{TL} = \{LT_{GP}, LT_{GM}, LT_{GF}\}$. Executing CDA we have the resulting terminology $\mathcal{T}$:

- $GP \equiv GM \sqcup GF$
- $GF \equiv GP \sqcap MALE$
- $GM \equiv GP \sqcap FEMALE$

Unfortunately, as the example shows, the terminology learned in this fashion may introduce a new problem, as cycles may be formed[3]. This goes against one of the general assumption we need to uphold—concept definitions are acyclic. We propose two approaches to deal with cycles: (i) restrict the set of examples added to $\mathcal{KB}$ and (ii) remove the cycles on a post-procedure.

[3] The definition for *GP* makes a cycle.

Algorithm 6 CDA with cycle removal main algorithm

Require: A terminology learning task $\mathcal{TL} = \langle \{LT_{A_1}, \ldots, LT_{A_m}\}, \mathcal{KB} = \langle TBox, ABox, RBox \rangle \rangle$.

Ensure: A Terminology $\mathcal{T}$

1: $\mathcal{T} \leftarrow \emptyset$
2: **for** each $LT_{A_i} \in \mathcal{TL}$ **do**
3: $\quad ABox' \leftarrow ABox$
4: $\quad$ **for** each $LT_{A_j} = \langle A_j, \mathcal{E}^+_{A_j}, \mathcal{E}^-_{A_j} \rangle \in \mathcal{TL}$ **do**
5: $\quad\quad$ **if** $i \neq j$ **then**
6: $\quad\quad\quad ABox' \leftarrow ABox' \cup \mathcal{E}^+_{A_j} \cup \mathcal{E}^-_{A_j}$
7: $\quad \mathcal{KB}' \leftarrow TBox, ABox', RBox$
8: $\quad$ Learn a description C_i for A_i using $\mathcal{KB}'$ and $\mathcal{E}^+_{A_i}, \mathcal{E}^-_{A_i}$
9: $\quad \mathcal{T} \leftarrow \mathcal{T} \cup A_i \equiv C_i$
10: $\mathcal{T} \leftarrow$ call Algorithm 7($\mathcal{TL}, \mathcal{T}$)
11: **return** $\mathcal{T}$

(i) If a concept $A_j \in \mathcal{TL} \setminus LT_{A_i}$, $i \neq j$, already has a definition and it uses A_i do not include A_i or A_j in $\mathcal{KB}$. This prevents that any new definition creates a cycle. As a consequence, an order for the learning process may be necessary. In this case, the best acyclic solution is not guaranteed.
(ii) After the end of the learning phase, to identify the concepts that incurred in cycles and relearn them, avoiding the current or a recurring cycle. Choose the definitions that yield smaller definitions.

The second method has a better chance to return the optimal acyclic solution, but it may relearn several concepts. In both methods, the best acyclic solutions is not guaranteed.

3.2.1 CDA with cycle removal

The first proposed method only deals with cycles with depth[4] of one, as exemplified in Example 3.

Example 3. The terminology containing the concepts: $A \equiv B_1 \sqcap B_2 \sqcap B_3$ and $B_1 \equiv B_4 \sqcap B_3 \sqcup A$ has a cycle with depth of one, because the concept A appears in the definition of B_1 while it also uses B_1 within its own description description, needing only one inferential step before incurring in a cycle.

Algorithm 6 is the entry point for the learning process. It runs the CDA for the terminology learning task and call the Algorithm 7 to remove the cycles of the learned terminology.

Algorithm 7 is used to remove the cycles of a set of concept learning tasks. It starts by defining an empty map of blocked concepts. A concept A_j is a blocked concept for the learning of a concept A_i, if in a previous iteration there was a cycle between A_i and A_j and A_i's definition was altered. Furthermore, it creates a set of concepts that are in need of cycle verification, beginning with all the concept learning tasks.

While there are concepts to check for cycles, it chooses two concepts within this set to verify if there is a cycle involving them. If a cycle exists between two concepts, it creates a new $\mathcal{KB}$ using the existing $\mathcal{KB}$ together with the set of concepts not blocked (lines 10, 11). All the dependents of a blocked concept are themselves blocked concepts. Finally, alternative definitions are learned for both concepts using their 'new' $\mathcal{KB}$s.

Once the alternative definitions are acquired, one must choose which of the two concepts will receive the alternative definition (line 14). The choice is made in favour of the alternative definition with the

[4] Depth is the number of inferential steps taken until a cycle is found.

Algorithm 7 Algorithm for removing cycles from the learning tasks

Require: A terminology learning task $\mathcal{TL}$, a terminology $\mathcal{T}$ possibly with cycles.
Ensure: A terminology $\mathcal{T}$

```
1: mapBlockedConcepts ← ∅ Contains the set of concepts blocked from the learning of each
   concept
2: conceptsToVerify ← T
3: while conceptsToVerify ≠ ∅ do
4:     Def_i ← conceptsToVerify.pop ; Def_j ← null;
5:     for each Def_p ∈ conceptsToVerify do
6:         if There is a cycle between Def_i and Def_p then
7:             Def_j ← Def_p
8:             break from the for loop
9:     if Def_j ≠ null then
10:        KB_Ai ← TL/{Def_j∪ DepEndents of Def_j∪ mapBlockedConcepts of Def_i}
11:        KB_Aj ← TL/{Def_i∪ DepEndents of Def_i∪ mapBlockedConcepts of Def_j}
12:        Learn a description C_i For A_i, using any algorithm that learns concepts in DL with KB_Ai
13:        Learn a description C_j For A_j, using any algorithm that learns concepts in DL with KB_Aj
14:        A_b ← A_i or A_j by the best description between C_i and C_j
15:        add A_b to conceptsToVerify
16:        add the other concept to the blocked concepts of A_b
17:        return T
```

smallest impact on the terminology, that is, the one with least difference in length with regard to the original definition. Lastly, the concept learning task whose definition was altered has to be verified, because new cycles may have been introduced, and the other concept learning task is added to alter concept's blocked set.

3.3 Finding dependencies on an existing terminology (CDT)

Finally, we devise a method that is able to find dependencies among concepts even in a ready-to-use terminology. It aims at compactness, since it finds parts of a concept description that can be replaced by a single concept. It is motivated by the fact that if part of the description has a semantically equivalent but syntactically smaller counterpart, then a substitution could take place generating a smaller though equivalent definition. Henceforth, we call this process *syntactic compression*.

3.3.1 Syntactic compression using a tree representation

In order to achieve a syntactic compression, it is necessary to find viable substitutions, that is, parts of a concept definition that encompass another concept.

At a first glance, the syntactical compression may be done performing a simple matching of the two logical definitions. However, this approach is very likely to not produce all possible substitutions, since most of the DL's constructors are commutable, as shown in Example 4.

Example 4. The logical definition $A \sqcap B_1 \sqcap B_2$ is equivalent with any conjunction of permutations of the concepts $\{A, B_1, B_2\}$, such as: (i) $A \sqcap B_2 \sqcap B_1$, (ii) $B_2 \sqcap A \sqcap B_1$, ...

In order to overcome this limitation, while still performing a syntactic compression, we propose the use of a n-tree representation of the concepts definitions, whose (i) internal nodes represent a constructor, (ii) leaves represent concepts and (iii) all the children of a constructor node are commutable. Allowing

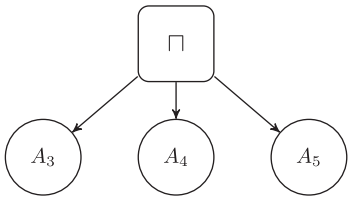


Figure 3 Example 5's syntactic tree representation

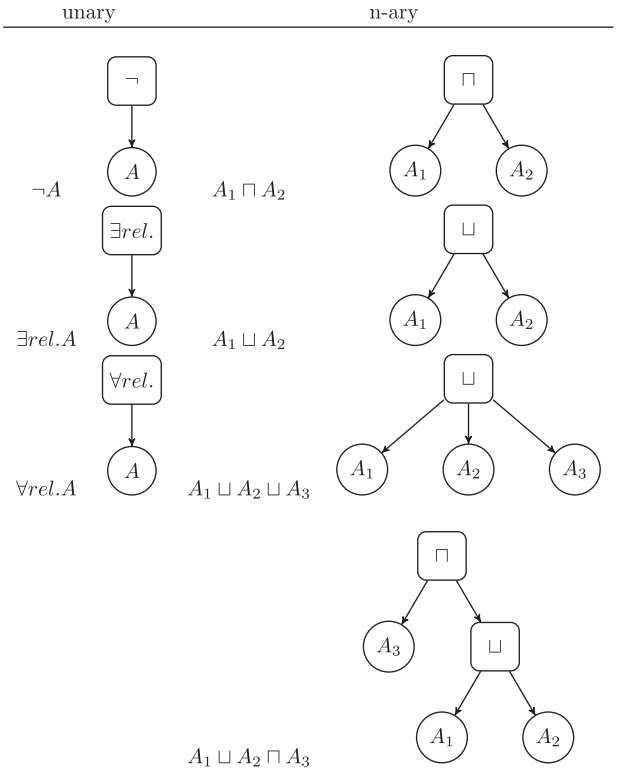


Figure 4 The syntactical tree representation of $\mathcal{ALC}$'s constructors

the commutation of the constructor's node children is essential, since identical definitions, unless for the order of literals, should have the same associated tree. This is illustrated in Example 5.

Example 5. Let A_1, A_2, A_3, A_4, A_5 concepts in a terminology, such that concept $A_1 \equiv A_3 \sqcap A_4 \sqcap A_5$ and $A_2 \equiv A_5 \sqcap A_3 \sqcap A_4$. Figure 3 shows the representation for both concepts:

Figure 4 shows all the correlations between $\mathcal{ALC}$'s constructors and their tree representation. We stress that the proposed method is language-independent as long as it is possible to separate the constructors of the DL language between unary and binary (commutable n-ary) constructors.

Removal of redundant descriptions. Algorithm 8 presents our procedure to syntactically compress a terminology. It receives as input a terminology and the set of available constructors, divided into unary and n-ary, and returns (possibly) compressed terminology. It starts by sorting the set of concepts in an ascending order of length, as a concept can only have a substitution involving a smaller concept. Next, for each concept in the terminology, it finds its corresponding tree form (line 4).

Once it has the tree representation with all concepts, it can proceed to find suitable substitutions. It does so by exhausting all the pairs of concepts with a smaller concept and a larger one (line 5), and calling Algorithm 11 (line 6) to find a proper substitution. After all the substitutions are made, it changes

Algorithm 8 Syntactical compression main algorithm

Require: a set of concept definitions **DEF** $= \{A_1 \equiv C_1, \ldots, A_m \equiv C_m\}$ and the set of valid constructors in the DL

Ensure: a set of compressed concepts definitions **DEF'** $= \{A_1 \equiv C'_1, \ldots, A_m \equiv C'_m\}$

1: sort **DEF** by the length of each $C_1, \ldots C_m$;
2: $\mathcal{T}_c \leftarrow \{\}$
3: **for** each $C_i \in$ **DEF do**
4: $\quad \mathcal{T}_c$.*add* Algorithm 9(C_i, 0, C_i length, unary constructors, unary constructors);
5: **for** each pair $\mathcal{T}_{ci}$ and $\mathcal{T}_{cj}$ in $\mathcal{T}_c$, where $i < j$ **do**
6: $\quad$ Algorithm 11($\mathcal{T}_{ci}$, $\mathcal{T}_{cj}$);
7: **for** each $\mathcal{T}_{ci} \in \mathcal{T}_c$ **do**
8: $\quad C_i \leftarrow$ TreeToDefinition($\mathcal{T}_{ci}$);
9: **return DEF'**

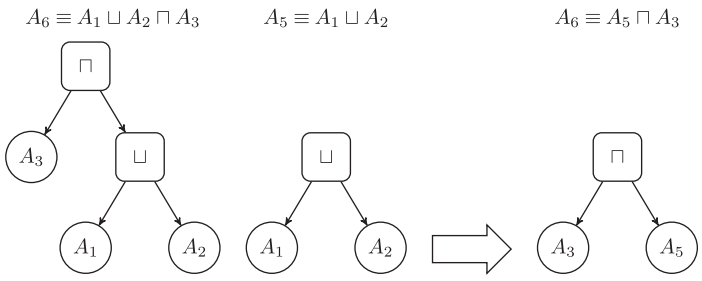
the modified original concept definitions by translating the tree representation back to its logical form (line 8). The translation method is done via a pre-order traversal in the tree.

Syntactic tree construction. Algorithms 9 and 10 are used to parse the logical syntax into the tree representation. In case all the n-ary constructors have the same precedence, as it is the case of $\mathcal{ALC}$ language, we still need a way to distinguish them in the tree. We do so by creating a hierarchy in the tree with them. Thus, the line 16 of algorithm 9 creates a subtree as soon as it identifies a constructor different from the previous one. In case the DL language defines a different precedence rule than $\mathcal{ALC}$, the algorithm must be changed to attend such a rule.

In the syntactic representation, a block is a part of the definition enclosed within parenthesis, and it may contain other blocks. The upper and lower bound are the begin and end of a concept or block, respectively.

Finding proper substitutions. Algorithm 11 is used to perform substitutions between two trees, a larger tree and a smaller tree. To perform a substitution, it must find a node within the larger tree that is the root of the smaller tree (line 4). In this representation, a larger tree contains a subtree equivalent to a smaller tree when: (i) all the internal nodes in the smaller tree have an equivalent node in the larger tree (also with equivalent arcs), and (ii) the subtree root of the larger tree contains all the nodes below the smaller tree root. Example 6 shows why the second constraint does not state that the larger tree subtree's root should be equal to the smaller tree's root. The problem of finding an equivalent tree is also known in the literature as *tree isomorphism* (Zemlyachenko *et al.*, 1985).

Example 6. Let $A_6 \equiv A_1 \sqcup A_2 \sqcap A_3$ and $A_5 \equiv A_1 \sqcup A_2$. Executing the substitution on the tree representation of A_6 and A_5 we have



Algorithm 9 Converting a logical description to a syntactical tree

Require: a concept description C; a lower bound LB; an upper bound UB; a set of unary constructors $S_u = U_C1, \ldots, U_Cn$ and a set of nary constructors $S_n = N_C1, \ldots, N_Cn$
Ensure: A syntactical Tree *root*

1: *root* ← ∅; *firstConcept* ← ∅; *index* ← *LB*;
2: **while** *index* ≤ *UB* **do**
3: *currentConstruct* ← *C*[*index*];
4: **if** *currentConstruct* is a block **then**
5: *aux* ←Algorithm 9(*C*, index, End block index, S_u, S_n);
6: **if** *root* is empty **then**
7: *root* ← *result*;
8: **else**
9: merge *root* with *result*;
10: increment index and continue to next loop iteration;
11: **if** *currentConstruct* ∈ S_n **then**
12: **if** *root* is empty **then**
13: *root* ← *currentConstruct*;
14: **if** *firstConcept* is not empty **then**
15: add *firstConcept* as a child of *currentConstruct*;
16: **if** *root* ≠ *currentConstruct* **then**
17: add *root* as a child of *currentConstruct*;
18: *root* ← *currentConstruct*;
19: increment index and continue to next loop iteration;
20: **if** *currentConstruct* ∈ S_u **then**
21: *currentConstruct* ← Algorithm 10;
22: *index* ← End index returned in line 21;
23: **if** *root* is empty **then**
24: *firstConcept* ← *currentConstruct*;
25: **else**
26: add *currentConstruct* as a child of *root*;
27: **if** *root* is empty AND *firstConcept* is not empty **then**
28: *root* ← *firstConcept*;
29: **return** *root*

4 Empirical evaluation

This section presents the experimental results aimed at validating the three MCL strategies devised in this work. First, we describe the datasets used in the experiments. Next, we present the experiments performed to validate each method individually. Finally, we describe the experiments that compare the proposed methods among each other and to a single concept learning system.

4.1 Datasets

The domains considered in this work are as follows:

- **The Kinship domain** defines family relations through a terminology containing three roles, *sibling*, *married* and *parent*, and two atomic concepts, *male* and *female*. The concepts that should be learned are *Grand Parent (GP), Grand Father (GF), Grand Mother (GM)*, according to their standard semantic.

Algorithm 10 Recursive method for unary constructors

Require: a concept description C; a lower bound LB; a set of unary constructors $S_u = U_C1, \ldots, U_Cn$ and a set of nary constructors $S_n = N_C1, \ldots, N_Cn$
Ensure: A syntactical Tree *node* and the ending index

```
1: index ← LB;
2: currentConstruct ← C[index];
3: if currentConstruct ∈ S_u then
4:     if currentConstruct is a unary quantIfiers then
5:         currentConstruct ← all items from index to relation Statement;
6:     currentConstruct.descendent ← Algorithm 10;
7:     index ← Algorithm 10 index;
8: else
9:     if currentConstruct is a blockthen
10:        currentConstruct ← Algorithm 9 (C, index, end block index, S_u, S_n);
11:        index ← end block's index;
12: return currentConstruct and index
```

Algorithm 11 Syntactical compression's substitution

Require: the larger tree $\mathcal{TR}_l$; the smaller tree $\mathcal{TR}_s$

```
1: nodesToVerify ← {root of TR_l};
2: while nodesToVerify is not empty do
3:     rootLarger ← nodesToVerify.pop;
4:     if the subtree rooted at rootLarger is equal to TR_s then
5:         replace the subtree rooted at rootlarger from TR_l by a node with the concept represented by TR_s
6:     else
7:         add children of rootLarger in nodesToVerify;
8: Return TR_s
```

- **Moral reasoner** (Wogulis, 1994): this dataset describes moral statements such as 'Someone is guilty of an action if it is blameworthy or he/she takes the blame for someone else'. It has a terminology composed of 25 atomic concepts and 20 non-primitive concepts. The ABox has assertions for 202 individuals. However, not all the assertions about non-primitive concepts are represented, in these cases it is necessary to infer them. Two experiments were done with this dataset. In the first one **(Exp1)**, the concepts that must be learned are *{guilty, blameworthy, responsible, notaccident}*, and in the second one, **(Exp2)**, *{guilty, blameworthy, responsible, notaccident, weak_intend, intend_c}*, increasing the learning process complexity with two more concepts. The concepts for each set of problems were chosen because of their inter-dependencies.
- **OAEI campaign**: different from CDA and CDE, CDT method can deal with terminologies that do not have individuals (ABox) because it works only with concept's descriptions (TBox). Because of that, we experiment CDT using a number of datasets from OAEI[5] campaign of ontology matching and all datasets packaged as examples on DL-Learner system[6]. In total, there were 46 datasets with a wide range of domains, for example, the kinship, moral, biomedical, and sizes, ranging from 0 to 10 480 complex concepts.

[5] http://oaei.ontologymatching.org/.
[6] http://dl-learner.org/Projects/DLLearner.

Table 2 CDE: Best threshold experiment. Hits indicate the number of found dependencies that matched those presented in the ground truth

Exp.	Threshold	# Pred.	# Relations Found	# Hits	Hits over Pred.
exp1	(0.8, 0.6, 0.4, 0.2, 0.1)	6	4	4	0.67
	0.9	3	2	2	0.67
exp2	(0.4, 0.2, 0.1)	15	4	4	0.27
	0.6	13	4	4	0.31
	0.8	9	4	3	0.33
	0.9	4	5	2	0.5

Table 3 CDA's validation experiment results

Experiment	Concept	Definition	Length
CL	GP	EXISTS parent.EXISTS parent.TOP.	5
	GF	(male AND EXISTS parent.EXISTS parent.TOP).	7
	GM	(female AND EXISTS parent.EXISTS parent.TOP).	7
CDA	GP	(grandfather OR grandmother).	3
	GF	(grandparent AND male).	3
	GM	(grandparent AND female).	3

4.2 Individually experimenting each method

In this section, we experimentally evaluated each of the three approaches.

4.2.1 CDE: Finding the best threshold value

CDE uses a threshold parameter to define a lower limit for the proportion of intersection of the examples associated with two concepts, after which the concepts are considered dependent by the method. In Melo *et al*. (2013a), the threshold value was arbitrarily set to 80%. Here, we empirically experiment on several values to decide which one is the best for the threshold parameter.

Thus, to analyse the behaviour of CDE approach according to the threshold parameter, we set it to 20%, 40%, 60%, 80% and 90% and compute the matching relationships using the *moral reasoner dataset*. Table 2 presents the average predictive results. The second column presents the threshold's value, aggregating them in the cases where the corresponding results are the same. The third column shows the number of predictions returned by the induced taxonomy and the fourth column shows the subset size of relationships that were indeed considered in the learned terminology. The fifth column shows the number of found dependencies that matched those presented in the target description, we called this metric as *Hits*. The results indicate that indeed the 80% threshold is a good value, followed closely by 90%. As it should be expected, requiring only a low intersection between two sets of examples yields a high number of predictions that are not actually established during the learning phase.

4.2.2 Experimenting CDA

CDA adds the learning examples to the background knowledge in order to induce the shortest solution, in terms of the length of the description, for a single concept learning task. We devised an experiment using the Kinship domain in order to validate this claim. In this experiment, we do not uphold the assumption that the terminology must be acyclic, since we would like to know if CDA's surrogates are indeed correct and can yield the shortest description for all the concepts in a terminology learning task. Table 3

Table 4 CDT: Results of the validation experiment, showing only the datasets that a different terminology was yielded

Dataset	#Concepts	Original size	Post size	Exec time in secs.
OAEI—conf_Cocus	5	19	**17**	0.005727
OAEI—NCI_small_overlapping_snomed	6851	74 191	**74 183**	684.26991
OAEI|NCI_while_ontology	10 280	151 316	**151 308**	1997.30955

shows the result for this experiment contrasting with the result achieved with the concept learning (CL) method.

CDA achieves a terminology of length nine, while the single concept learning system DL_Learner found a terminology of length nineteen. Furthermore, CDA founds the shortest possible solution for all the three concepts in the terminology learning task.

4.2.3 Experimenting CDT

Different from the CDE and CDA methods, CDT can be used outside the terminology learning problem, because it is a method for theory restructuring. Table 4 presents the results obtained with CDT, but only to the cases that it was able to get some difference in the terminology.

The method achieved *syntactic compression* on only 3 of the 46 terminologies: *conf_Cocus*, *NCI_small_overlapping_fma.owl* and *NCI_whole_ontology.owl*. In the *conf_Cocus* domain, the original terminology is composed of five complex concepts (Administrator, Document, Event, Event_Setup and User), 'with Document' and 'Event' having the same definition: '$\exists created\_by.Person$'. The method arbitrarily substituted one of the definitions by the other concept, for example, returning the description $Event \equiv Document$. In this particular case, it might be argued that the real meaning is lost when this substitution is performed, but from a pure logical point of view the semantic is maintained.

The other remaining Biomedicine domains share much of the concepts and descriptions. In fact, the terminology of NCI_whole_ontology.owl contains the NCI_small_overlapping_fma.owl. Both had the same two substitutions: the concept 'Complement_Component-4' was substituted by its definition on concepts 'Complement_Component-3' and 'Complement_Component-5'. Both concepts where the substitution occurred have a structure of many conjunctions of existential restrictions. These particular replacements are difficult to visualise because of the size of the descriptions (41 and 57, respectively). This might suggest a direct relationship between a terminology size and its complexity and how difficult it is to find all the simplifications manually.

Regarding the cases where CDT made no difference, we observed two situations: (1) a number of these datasets have only atomic concepts and roles in the ABOX but no concepts definitions to compress and (2) in the other cases, the method had no impact on the size of the terminology, as they were already syntactically compressed. More than anything, this might be a testament of the quality of the terminologies.

4.3 Experiments comparing the three methods

In this section, we describe experiments conducted towards comparing CDE, CDA and CDT, among each other and to a standard single concept learning system, namely DL-Learner. It is necessary to define the knowledge base ($\mathcal{KB}$) and the sets of examples ($\mathcal{E}_{p_i}$, $\mathcal{E}_{n_i}$) for each concept learning A_{C_i}. The latter is performed either directly from the original examples or by performing inference along the terminology and the knowledge base. While the $\mathcal{KB}$ is the same in all A_{C_i}, $\mathcal{E}_{p_i}$ and $\mathcal{E}_{n_i}$ vary in accordance with the concept. The experiments are conducted on the Kinship and moral reasoner datasets, previously described.

Table 5 Toy example results: Concepts definitions and their sizes found with the methods CDE, CDA, CDT and DL-Learner (DL-L)

Approach	Concept	Definition	Size
DL-Learner	GP	EXISTS parent.EXISTS parent.TOP.	5
	GF	(male AND EXISTS parent.EXISTS parent.TOP).	7
	GM	(female AND EXISTS parent.EXISTS parent.TOP).	7
CDE	GP	EXISTS parent.EXISTS parent.TOP.	5
	GF	(grandparent AND male).	3
	GM	(grandparent AND female).	3
$CDA_{(1)}$	GP	(grandfather OR grandmother).	3
	GF	(male AND EXISTS married.grandmother).	5
	GM	(female AND EXISTS parent.EXISTS parent.TOP).	7
$CDA_{(2)}$	GP	EXISTS parent.EXISTS parent.TOP.	5
	GF	(grandparent AND male).	3
	GM	(grandparent AND female).	3
$CDA_{(3)}$	GP	EXISTS parent.EXISTS parent.TOP.	5
	GF	(grandparent AND male).	3
	GM	(grandparent AND female).	3
CDT	GP	EXISTS parent.EXISTS parent.TOP.	5
	GF	(grandparent AND male).	3
	GM	(grandparent AND female).	3

4.3.1 Datasets

Kinship dataset. Regarding this dataset, we want to investigate three issues: (i) whether the learned descriptions are according to the general semantics of the kinship domain; (ii) which approach yields the more readable and concise terminology and (iii) if the ordering of concepts defined beforehand brings any difference to the terminology learning task. Thus, we run and compare DL-Learner, the approach for single concept learning, CDT, CDE and CDA. The latter one considered three cases: (1) the order established by CDE's taxonomy, in this case $<$ GP, GF, GM $>$; (2) the reversed order of the one considered in (1) ($<$ GM, GF, GP $>$) and (3) removing cycles. The first two cases follow the definition of $\mathcal{TL}$ and avoid cycles.

Moral reasoner dataset. Concerning this dataset, we use a 10-fold cross-validation procedure to investigate the following issues:

(i) how the completeness of the examples over the original set of individuals influences the proposed approaches? Towards answering this question, we vary the training set considering 20%, 40%, 60%, 80% and 100% of examples.
(ii) what is the quality of the terminology learned by CDE, CDA and CDT, including a comparison with DL-Learner? We investigate this issue using quantitative and qualitative measures. For the former, we considered the F-measure while for the latter (1) the size, encompassing concepts, relations and constructors, and (2) the similarity to the original terminology. This last one we accomplished by the subsumption relation: (a) does the learned terminology subsumes the original one (more general)? (b) does the original terminology subsumes the learned one (more specific)? If both questions are true, then the learned and target terminology are equivalent (this would be the best result possible).

4.3.2 Experimental results

Kinship dataset. The results obtained considering the Kinship dataset are depicted in Table 5.

From that, it is possible to see that all the concepts description follow the correspondent semantics. Moreover, CDE, CDA and CDT were able to learn a more concise and readable terminology than DL-Learner. Additionally, in order to uphold DL's assumption of acyclic terminologies we used: a version of CDA that avoids cycles ((1) and (2)), in this case orderings are particularly important, that is, different orders entail different results; and a version of CDA that removes the cycles (3) (Algorithms 6 and 7). The results in Table 5 supports such conclusion and shows that CDA avoiding cycles is able to achieve the same terminology as CDE and CDT by reversing its learning order.

Moral reasoner dataset. In the rest of this section, we will refer to CDA avoiding cycles as CDA_1 or $CDA1$, and to CDA removing cycles as CDA_2 or $CDA2$.

Table 6 exhibits the results for all the proposed approaches (CDE, CDA1, CDA2 and CDT), considering the two experiments (Exp1 and Exp2) described in Section 4.3.1 and different percentages of the training examples. The proportion of positive and negative examples was maintained in each set. Regarding CDE, we present the results for three different values of the threshold parameter (60%, 80%, 90%). Moreover, to evaluate the approaches, we considered the percentage of original dependency among concepts found in the learned terminology (% Hits) and Table 6 depicts the average of the 10 folds.

Observing the results in Table 6, both CDE and CDA performed worse than expected. The CDA (1 and 2) behaviour could be explained by it being even more sensitive to completeness than we expected, since we direct 10% of the original examples to the evaluation fold in every scenario (so the real percentages of the original dataset are: 90%, 72%, 54%, 36% and 18%). To confirm this insight, we executed the same scenarios in the whole original dataset (without cross-validation).

Table 7 confirmed our intuition, showing that with the whole set of individuals CDA1 was able to find all 'hits' for Exp1, and even more: it is the only approach able to achieve this. Moreover, the results for Exp2 are also improved. For Exp1, CDA2 has a comparable result to the other approaches, but it is a lot better on Exp2, which might indicate that CDA2 behaves better on more complex terminology learning problems. The results for CDE did not change as much as when the completeness is relaxed. Therefore, when all the individuals are known or when the examples are a more significant sample of the domain, the CDA method seems the better choice. The CDT approach found far less dependencies than its counterparts. Table 8 shows the CDT's hits.

Regarding the analysis of the terminology quality, DL-Learner was also considered. Figure 5 shows the plots of f-measure results for each of the concepts and for all the scenarios. It is possible to notice that all the approaches performed statistically similar in both Exp1 and Exp2, except CDA in Exp2, when inducing the definition of *weak_intend*. However, it is important to notice that in this specific case, the appropriate relationships is not considered (see Table 6), possibly because the lack of full completeness due to the 10% of examples reserved for validation, as discussed previously. Therefore, terminology learned with MCL approaches achieve similar predictive results compared to the single concept learning approach. This partially validates the research question **Q1**: terminologies learned regarding dependencies among the concepts have comparable quality (predictive power) to its counterpart learned with no regard for the dependencies.

Regarding the qualitative metrics, we used the whole set of examples. For the size of the found terminologies, on the far left of Figure 6, we show the original size of the terminology, and none of the approaches achieve results equal or smaller than it. On the far right, we have the size of the terminologies without MCL. None of the approaches produced terminologies larger than it. The CDT results are equal to the DL-Learner's because no redundancy is found. Apart from CDT, all the proposed methods induced far smaller terminologies than the equivalent concept learning without addressing the dependencies.

In conjunction with the results related to the predictive power, we can conclude that if dependencies among the concept of the domain that need to be learned are found then *clearer* and *concise* terminologies are yield while their *quality* is maintained, successfully answering the research question **Q2**.

For the size of concept per experiment, see Figure 7. The three CDE strategies have very similar results, consistently better or equivalent to DL-Learner, except on 'responsible' in Exp1 and 'notaccident' in Exp2. CDA_1 has the best result for 'guilty' in Exp1, but its 'blameworthy' was the worst among our approaches, exp. Interestingly, both CDA found by far the best result for 'blameworthy' and 'intend_c'

Table 6 Aggregate results of relaxing completeness experiment for the methods CDE, CDA and CDT

Approach	Experiment	Percentage	% Hits
CDE–0.9	exp1	100%	33%
		(80%, 60%, 40%, 20%)	0%
	exp2	100%	60%
		80%	80%
		60%	60%
		40%	40%
		20%	20%
CDE–0.8	exp1	100%	66%
		(80%, 60%, 40%, 20%)	0%
	exp2	100%	20%
		80%	80%
		60%	60%
		40%	40%
		20%	20%
CDE–0.6	exp1	100%	66%
		80%	66%
		(60%, 40%, 20%)	0%
	exp2	100%	20%
		80%	20%
		60%	60%
		40%	40%
		20%	20%
CDA_1	exp1	100%	33%
		(80%, 60%, 40%, 20%)	0%
	exp2	(100%, 80%)	20%
		60%	0%
		(40%, 20%)	0%
CDA_2	exp1	100%	33%
		(80%, 60%, 40%, 20%)	0%
	exp2	(100%, 80%)	20%
		60%	0%
		(40%, 20%)	0%
CDT	exp1	(100%, 80%, 60%, 40%, 20%)	0%
	exp2	(100%, 80%, 60%, 40%, 20%)	0%

in Exp2, with CDA_2 being even better, while it is only average in the other results. There is a trade-off on the size of the concepts, as when some instances choose a smaller definition for a concept, the size of other related concepts are affected, because the latter might not be able to use the former. This explains why CDA_2 had the best solutions overall, because different from the other approaches it tries alternative definitions for both concepts when a dependency incurs in a cycle.

Table 7 Full completeness over the individuals experiment with the methods CDE, CDA and CDT

Approach	Experiment	# Dependencies	% Hits
CDE(0.9, 0.8)	exp1	4	66%
	exp2	4	20%
CDE—(0.6)	exp1	4	66%
	exp2	5	20%
CDA_1	exp1	3	100%
	exp2	6	40%
CDA_2	exp1	4	66%
	exp2	5	80%
CDT	exp1	0	0%
	exp2	0	0%

Table 8 Full completeness experiment CDT's hits

Cover	**Total**		**20%**		**40%**		**60%**		**80%**		**100%**	
	Exp		**Exp**		Exp		Exp		Exp		Exp	
Fold	1	2	1	2	1	2	1	2	1	2	1	2
–	0	0										
1			0	0	0	0	0	0	0	0	0	0
2			0	0	0	0	0	0	0	0	0	0
3			0	1	0	0	0	0	0	0	0	0
4			0	0	0	0	0	0	0	0	0	0
5			0	0	0	0	0	0	1	0	0	0
6			0	0	0	0	0	0	0	0	0	0
7			0	0	0	0	0	0	0	0	0	0
8			0	2	1	0	0	0	1	1	0	0
9			0	0	0	0	0	0	0	0	0	0
10			0	0	0	0	0	0	1	0	0	0

Figure 8 contrasts the size of concepts learned with their original definitions. The values are calculated through the formula $\frac{|original\ definition|}{|learned\ definition|} - 1$, where values over 0 indicates that learned terminology is smaller than the original one. Most of the results were negative, with the size being between 40% to 70% bigger, while some results are positively better as 'blameworthy' in Exp1 and 'responsible' and 'weak_intend' in Exp2, with definitions 200% smaller than the original ones.

Additionally, a particularly important result is the performance of CDA concerning concept 'blameworthy' in Exp2. It provides the higher decrease in size. Due to the fact that CDA uses assertions about all concepts being learned and 'blameworthy' is the concept that is most dependent on other concepts from the learning task, it is the one that most benefits from CDA approach. On the other hand, CDE is the worst choice for this concept, since the taxonomy is more complex.

Figure 9 uses the same formula, showing that CDEs and CDAs provide smaller definition than DL-Learner.

Furthermore, for Exp1 all the definitions learned subsumed the original definitions, while for Exp2 (i) definition for concepts {blameworthy, guilty, responsible} subsumes their original definition when DL-Learner and CDA are performed; (ii) definition for concepts {guilty, intend_c, weak_intend} subsumes their original definition when CDE is performed; (iii) the original definition for {notaccident} subsumes

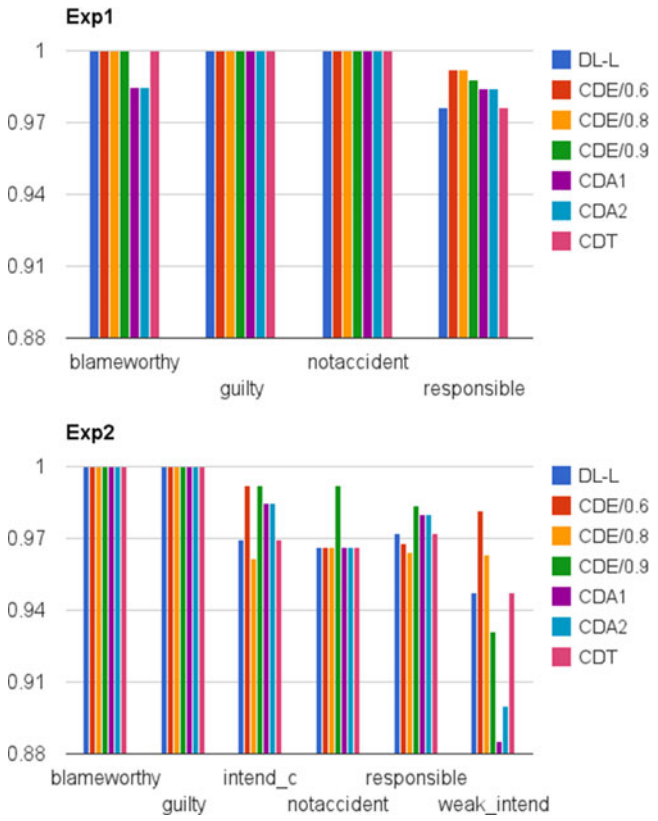


Figure 5 F-measure for Exp1 (top) and Exp2 (down) considering scenarios with CDE, CDA, CDT and DL-Learner (DL-L)

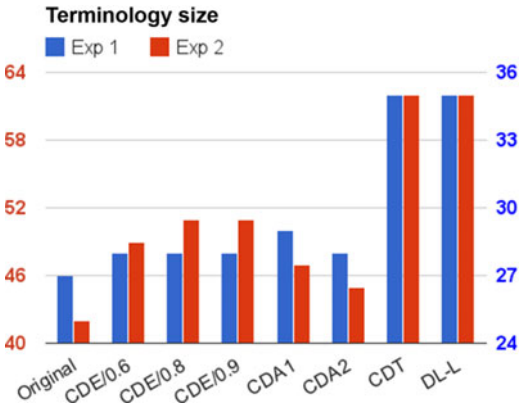


Figure 6 Size of found terminology for Exp1 and Exp2 with methods CDE, CDA, CDT and DL-Learner (DL-L)

the definition learned with DL-Learner and (iv) the original definition for {notaccident, responsible} subsumes the learned definition when running CDA and CDE.

Summarising, these results show the potential of the MCL approaches proposed in this work, to find smaller more readable solutions and, in some cases, solutions equivalent to those created by human beings. Furthermore, CDE proved to be less susceptible to the completeness assumption, while CDA was very sensible to it. CDA_2 displayed a very good performance, being the closest to the original terminology on the more complex experiment (Exp2); however, one must not forget the additional cost of relearning the concepts to remove the cycles. Finally, the CDT approach found far less dependencies than its counterparts. This is due to the fact that syntactical substitutions are far more unlikely when the set

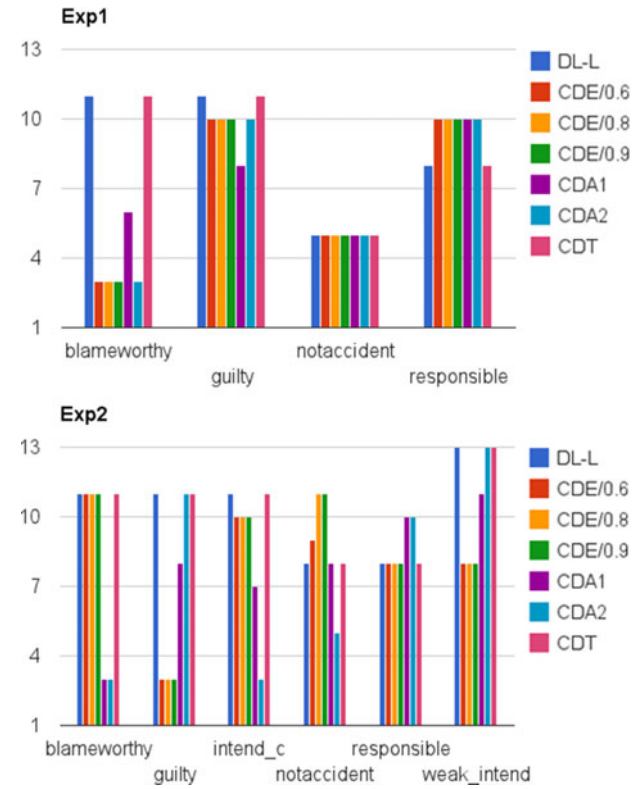


Figure 7 Size of concepts definitions for Exp1 (top) and Exp2 (down) with methods CDE, CDA, CDT and DL-Learner (DL-L)

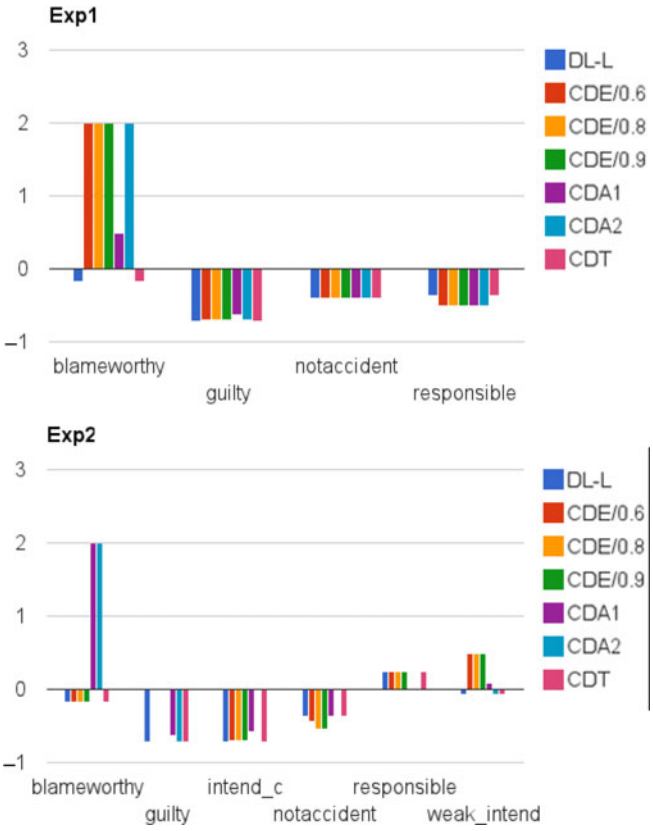


Figure 8 Size difference for Exp1 (top) and Exp2 (down) w.r.t. Original Definition with methods CDE, CDA, CDT and DL-Learner (DL-L)

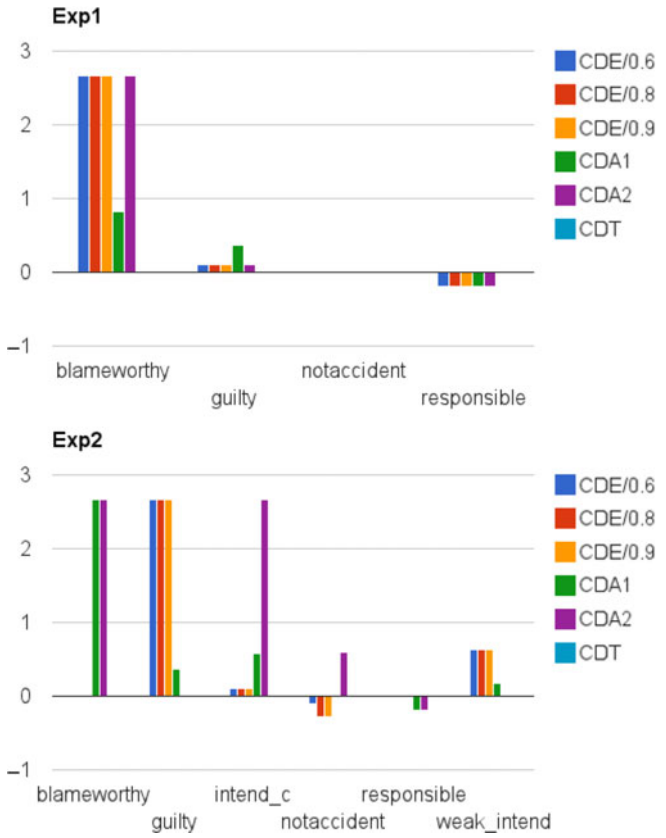


Figure 9 Size difference for Exp1 (top) and Exp2 (down) w.r.t. DL-Learner (DL-L) with methods CDE, CDA and CDT

of concept learning problems have a wide range of possible solutions. Thus, we can answer the research question **Q3**, that is, depending on the employed method, we might well have more concise solutions.

5 Related works

Related work can be divided into three major categories. First, we briefly discuss works more related to ours, which are the ones that also focus on concept learning using DLs. Next, we discuss the works that also focus on learning multiple concepts, but using logic programming as the Knowledge Representation Language. Finally, we briefly discuss the more general approach of learning multiple yet related tasks.

5.1 Concept learning with DLs

In this case, we can see related work from three different points of view: (1) previous work regarding MCL in a DLs setting; (2) works related to single concept learning in DLs, as such methods are used to learn the concept definitions once one of the proposed methods is applied and (3) previous work that learn multiple concepts using logic programming as the underlying representation language, in the field of inductive logic programming.

Focusing on the single concept learning task, most work induce terminologies either independently or using an ad hoc order (Iannone *et al.*, 2007; Fanizzi *et al.*, 2008; Lehmann, 2009). These works are necessary to the MCL methods presented here because an integral part of the task involves learning a single concept definition[7], which is learned using one of the existing methods. Other previous works focus on divide-to-conquer strategies to learn single concepts by combining partial solutions built either from a subset of the positive and negative examples, or from different data sources (Tran *et al.*, 2017;

[7] DL-Learner was used.

Gao *et al*., 2018; Konys, 2018). In a future work, a study of the underlying bias of this part of the process should be done, because it appears that each method has a slightly different focus, either on the general to specific direction (Fanizzi *et al*., 2008; Lehmann & Hitzler, 2010) or in the specific to general (Iannone *et al*., 2007). To that and to better validate our method, we can also leverage recently proposed logical measures to the problem of ontology leaning (Shamsfard & Barforush, 2003; Barforush & Rahnama, 2012), such as dissimilarity and complexity (Sazonau & Sattler, 2017). YinYang (Iannone *et al*., 2007) provides a method that refines definitions after a single concept is learned, which is related to our *CDT* method. However, while the refinement focuses on a single concept, *CDT* tries to find dependencies within a whole terminology and refine it, from such dependencies.

Next, with respect to terminology learning considering the induction of a set of related but distinct concepts, to the best of our knowledge, the literature is rather scarce. Previous works have tried to handle this problem by including the human in the loop: in Konev *et al*. (2017b), taxonomies containing possibly related concepts are learned by focusing on the communication process between the ontology engineer and a domain expert, while in Sazonau *et al*. (2015) the terminology induction is targeted as a multi-objective problem, where the induced hypotheses are presented to a domain expert to accept or reject them, aiming at quality measures. Here, our three different approaches lean terminologies completely automatically. Another work related to ours conceives the analysis of formal concepts to find out the best way to learn a terminology (Baader *et al*., 2007). One seemingly related work is Distel (2011), where definitions for multiple concepts are induced using formal concept analysis concepts. However, this work does not do MCL as described here, it is more in-line with finding the complete set of relationships within a terminology without the use of new examples.

5.2 Multiple predicate learning in ILP

Finally, a large portion of the theoretical background used by concept learning methods in DLs can be traced back to earlier works on the ILP area, specially with regard to some of the basic formalisms. The seminal works (Muggleton, 1991, 1992) provide some of the shared formalisms of this method. Although, a big theoretical intersection exists between the two fields, specific implementations are required because of particular idiosyncrasies that emerge from the particular knowledge representation—descriptions logics and logic programming, in concept learning in DL and predicate learning in ILP, respectively.

Although a direct correlation between solutions for ILP and concept learning might not occur, the cross-pollination of ideas is healthy for both areas of research. In De Raedt *et al*. (1993), the problem of multiple predicate learning is described, the description is very similar to Definition 3. It presents the major drawbacks and some path to solutions for multiple predicate learning. In essence, this work states that learning multiple predicates is more difficult than learning single predicates, as the former is not limited to the generation of several independent predicate definitions but involves the discovery of predicates dependencies. One of the possible solutions for the problem is the proper ordering of the predicate learning tasks, we presented in this paper Section 3.1.2 a method that aims at that. A different solution called MPL is presented, in it the refinement operator used deals with clauses with only the body, assigning a predicate to the head as the last step when evaluating a clause. This allows for evaluating a clause definition apart from a particular predicate. In this instance, the predicates are learned out of order and might contain multiple clauses with other predicates that were also learned in the body. As it is, MPL would not work in the DL setting because one of the basic assumptions is that a concept has only one definition, with some changes this might work within the assumptions, if instead of working with clauses it used hypothesis.

The mechanisms proposed in the literature to solve multiple predicates learning can be separated into approaches that work in the normal ILP with some kind of interleaving learning of clauses (De Raedt *et al*., 1993) setting or approaches that use an alternative setting to minimise some of the problems (De Raedt & Lavrač 1996; Muggleton & Bryant, 2000). These types of solutions cannot be directly applied to the problem in DL because in logic programming a predicate (concept) can be defined by several clauses, while in DL a concept (predicate) can have at most one definition. Bratko (1999) presents an ad hoc

method that have profound similarities with DL, and it generates full hypothesis instead of doing one clause of a time.

5.3 Multitask and curriculum learning

In a more general point of view, the algorithms developed here are related to the multitask learning area (Caruana, 1997), which also focuses on jointly learning multiple related tasks, with a number of successful applications (Liu *et al*., 2015, 2016). Thus, in case, one models such tasks using a concept learning setting, the algorithms developed here could also be applied to tackle the same type of problems handled by that area.

In addition, there is a constant effort on characterising and developing curriculum learning algorithms, so that machines would learn in an organised way (Bengio *et al*., 2009; Jiang *et al*., 2015). To achieve that, different concepts could be introduced at different times, in the same way that most of human educational systems work. The basic idea is to start from easier tasks and gradually moving on to the more difficult ones. The algorithms developed here aim at this same strategy, however addressing concept learning in a DL setting.

6 Conclusions

This paper presented three strategies for terminology learning of multiple (related) concepts in the DL setting. The first one, named CDE, automatically defines an order for learning the concepts, where this order is represented by a taxonomy tree of the concepts. This is accomplished by analysing the taxonomic intersection of the examples sets associated with each concept. The second one, named CDA, does not require such an order for learning the collection of concepts. Instead, it assumes temporary assertions for all the other concepts that are not selected to be learned. As the concepts may be considered to be included in each other definition, we developed two strategies to not include cyclic definitions in the final terminology. The third method, name CDT, is applied after the terminology learning task is concluded. It substitutes a subset $\boldsymbol{X}$ of the terminology by another concept c, if $\boldsymbol{X}$ is syntactically equivalent to the definition of c. This aims at syntactical compression while maintaining the original semantics. In all the cases, we expect that the returned terminologies are concise and readable descriptions of the concepts included in the domain of discourse.

We validated the proposed methods by examining results yielded from two domains: a toy problem from the kinship domain and a more elaborated problem, denominated ‘moral reasoner’ (Lehmann & Hitzler, 2010). Both of them are freely available in the DL-Learner package. From the first one, it was possible to observe that the terminologies learned with CDE and CDA were more concise than the one learned with DL-Learner (the baseline single concept learning system). Moreover, CDA was more versatile than CDE. Analysing the second domain, it was possible to observe that the threshold parameter required to define a proper relationship in CDE method gets more sensitive as the task is more complex. With a relatively simple task, varying this parameter did not cause any significant difference in the results.

The completeness of individuals interferes in both CDE and CDA, although the later is more sensible to this feature. For example, by varying the size of the training set, and as a consequence, the representativeness of the original individuals in the examples set of each concept, terminologies with different qualities were yielded by both methods. As both methods strongly rely on the set of examples to make crucial decisions, the closer from the original individuals are the assertions, the better the approaches will behave.

Finally, the quantitative scores are only slightly affected by the different learning process and parameters. On the other hand, with the best selected threshold parameter and the closest set of individuals to the original task, all the methods, with exception of CDT, were able to fetch more concise and readable terminologies than the SCL and CDA strategies outperformed all the other methods. CDT found far less dependencies than its counterparts. This is due to the fact that syntactical substitutions are more unlikely when the set of concept learning problems has a wide range of possible solutions, which was the case of the terminologies learned by the baseline SCL method.

In addition, in order to validate CDT for syntactic compression, we used a set of 46 datasets. Among the datasets were some particularly large terminologies, one of them with over 10 000 concepts. We then went on to verify the relationship between the execution time and the terminology complexity. Analysing the results, we concluded that the relationship is exponential but, as it seems, only prohibiting on very large terminologies (tens of thousands range). In terminologies with less than 2000 concepts, the runtime was below the 1-minute mark.

In a future work, we would like to establish a formalism so that ontology engineers could evaluate an automatically learned terminology through queries (Konev *et al*., 2016) and point out where it could be changed to be better understood, in a ontology revision loop (Nikitina *et al*., 2012; Dong *et al*., 2017). Furthermore, we believe that the proposed approach would bring significant benefits when learning *probabilistic description logics* (Revoredo *et al*., 2011; Gutiérrez-Basulto *et al*., 2017), since the probability values founded are dependent on the pre-existing values; therefore, defining it in a better order may yield better results. Finally, in this manuscript, we focused on the size of the terminology as the quality measure. However, the similarity among a human-written terminology, a terminology with independent concepts, and the terminologies found by our methods would also be a promising way of contemplating the benefits of learning the concepts jointly. There are previous papers that propose similarity metrics among DL ontologies (Maedche & Staab, 2002; Janowicz & Wilkes, 2009; Sánchez-Ruiz *et al*., 2011), and we intend to investigate their use within our methods in the future.

Acknowledgements

The first author would like to thank CAPES for the financial support through a master scholarship. The second author was partially supported by Capes-DAAD, and the third author would like to thank the Brazilian research agencies CNPq and FAPERJ for the financial support.

References

Antoniou, G. & Van Harmelen, F. 2004. Web ontology language: Owl. In *Handbook on Ontologies*, 67–92. Springer.

Baader, F. & Nutt, W. May 2010. Basic description logics. In *The Description Logic Handbook*, Baader, F., McGuinness, D. L., Nardi, D. & Patel-Schneider, P. F. (eds), 2nd edition. Cambridge University Press, 47–100.

Baader, F., Ganter, B., Sertkaya, B. & Sattler, U. 2007. Completing description logic knowledge bases using formal concept analysis. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI) 2007*, **7**, 230–235.

Barforush, A. A. & Rahnama, A. 2012. Ontology learning: revisted. *Journal of Web Engineering* **11**(4), 269–289.

Bengio, Y., Louradour, J., Collobert, R. & Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 41–48.

Berners-Lee, T., Hendler, J. & Lassila, O. 2001. The semantic web. *Scientific American* **5**(284), 34.

Brachman, R. J. & Levesque, H. J. 2004. *Knowledge Representation and Reasoning*, 1st edition. Elsevier.

Bratko, I. 1999. Refining complete hypotheses in ILP. In *Inductive Logic Programming*, Dzeroski, S. & Flach, P. (eds.) 1634. *LNCS.* Springer Berlin Heidelberg, 44–55.

Bühmann, L., Lehmann, J. & Westphal, P. 2016. Dl-learner - A framework for inductive learning on the semantic web. *Journal of Web Semantics* **39**, 15–24.

Caruana, R. 1997. Multitask learning. *Machine learning* **28**(1), 41–75.

De Raedt, L. 2008. *Logical and Relational Learning*. Springer.

De Raedt, L. & Lavrač. N, 1996. Multiple predicate learning in two inductive logic programming settings. *Logic Journal of IGPL* **4**(2), 227–254.

De Raedt, L., Lavrac, N. & Dzeroski, S. 1993. Multiple predicate learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1037–1042.

Distel, F. 2011. *Learning Description Logic Knowledge Bases from Data Using Methods from Formal Concept Analysis*. PhD thesis, TU Dresden.

Domingos, P. 1998. Occam's two razors: The sharp and the blunt. *In KDD*, 37–43.

Dong, T., Duc, C. L. & Lamolle, M. 2017. Tableau-based revision for expressive description logics with individuals. *Journal of Web Semantics* **45**, 63–79.

Fanizzi, N., d'Amato, C. & Esposito, F. 2008. Dl-foil concept learning in description logics. In *Inductive Logic Programming*, **5194**. *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 107–121. ISBN 978-3-540-85927-7.

Funk, M., Jung, J. C., Lutz, C., Pulcini, H. & Wolter, F. July 2019. Learning description logic concepts: When can positive and negative examples be separated? In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-2019*, 1682–1688. International Joint Conferences on Artificial Intelligence Organization. doi: 10.24963/ijcai.2019/233. https://doi.org/10.24963/ijcai.2019/233.

Gao, W., Guirao, J. L., Basavanagoud, B. & Wu, J. 2018. Partial multi-dividing ontology learning algorithm. *Information Sciences* **467**, 35–58.

Gutiérrez-Basulto, V., Jung, J. C., Lutz, C. & Schröder, L. 2017. Probabilistic description logics for subjective uncertainty. *Journal of Artificial Intelligent Research* **58**, 1–66.

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P. F., Rudolph, S. *et al.* 2009. Owl 2 web ontology language primer. *W3C Recommendation* **27**(1), 123, 2009.

Iannone, L., Palmisano, I. & Fanizzi, N. 2007. An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence* **26**(2), 139–159.

Janowicz, K. & Wilkes, M. 2009. SIM-DLA: A novel semantic similarity measure for description logics reducing inter-concept to inter-instance similarity. In *European Semantic Web Conference*. Springer, 353–367.

Jiang, L., Meng, D., Zhao, Q., Shan, S. & Hauptmann, A. G. 2015. Self-paced curriculum learning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, 2694–2700.

Konev, B., Ozaki, A. & Wolter, F. 2016. A model for learning description logic ontologies based on exact learning. In *Proceedings of AAAI-16*.

Konev, B., Lutz, C., Ozaki, A. & Wolter, F. 2017a. Exact learning of lightweight description logic ontologies. *Journal of Machine Learning Research* **18**, 201:1–201:63.

Konev, B., Lutz, C., Ozaki, A. & Wolter, F. 2017b. Exact learning of lightweight description logic ontologies. *The Journal of Machine Learning Research* **18**(1), 7312–7374.

Konys, A. 2018. Knowledge systematization for ontology learning methods. *Procedia Computer Science* **126**, 2194–2207.

Krötzsch, M., Simancik, F. & Horrocks, I. 2012. A description logic primer. *CoRR*, abs/1201.4089. http://arxiv.org/abs/1201.4089.

Lehmann, J. 2009. DL-learner: Learning concepts in description logics. *Journal of Machine Learning Research* **10**, 2639–2642.

Lehmann, J. & Hitzler, P. 2010. Concept learning in description logics using refinement operators. *Machine Learning* **78**(1–2), 203–250. ISSN 0885-6125.

Lima, R., Espinasse, B. & Freitas, F. 2018. Ontoilper: an ontology-and inductive logic programming-based system to extract entities and relations from text. *Knowledge and Information Systems* **56**(1), 223–255.

Liu, Y., Nie, L., Han, L., Zhang, L. & Rosenblum, D. S. 2015. Action2activity: Recognizing complex activities from sensor data. *In IJCAI*, 1617–1623.

Liu, Y., Zheng, Y., Liang, Y., Liu, S. & Rosenblum, D. 2016. Urban water quality prediction based on multi-task multi-view learning. *In IJCAI*, 2576–2582.

Maedche, A. & Staab, S. 2001. Ontology learning for the semantic web. *IEEE Intelligent Systems and Their Applications* **16**(2), 72–79.

Maedche, A. & Staab, S. 2002. Measuring similarity between ontologies. In *International Conference on Knowledge Engineering and Knowledge Management*. Springer, 251–263.

Melo, R., Revoredo, K. & Paes, A. 2013a. Terminology learning through taxonomy discovery. In *2013 Brazilian Conference on Intelligent Systems*, 169–174.

Melo, R., Revoredo, K. & Paes, A. 2013b. Learning multiple description logics concepts. In *ILP 2013 Late Breaking Papers*, 17–22.

Melo, R., Revoredo, K. & Paes, A. 2014. Syntactic compression of description logics terminologies. In *Brazilian Conference on Intelligent Systems*, 180–185.

Mitchell, T. 1997. *Machine Learning*. McGraw-Hill, New York.

Muggleton, S. 1991. Inductive logic programming. *New Generation Computing* **8**(4), 295–318.

Muggleton, S. 1992. *Inductive Logic Programming*. Morgan Kaufmann, 1992.

Muggleton, S. H. & Bryant, C. H. 2000. Theory completion using inverse entailment. In *Inductive Logic Programming*. Springer, 130–146.

Nikitina, N., Rudolph, S. & Glimm, B. 2012. Interactive ontology revision. *Web Semantics: Science, Services and Agents on the World Wide Web* **12**, 118–130.

Ozaki, A. 2020. Learning description logic ontologies: Five approaches. where do they stand? *KI - Künstliche Intelligenz* 34, 317–-327.

Revoredo, K., Ochoa-Luna, J. & Cozman, F. 2011. Learning probabilistic description logics: A framework and algorithms. In *In proceedings of the MICAI*, **7094**. *LNCS*. Springer, 28–39.

Sánchez-Ruiz, A. A., Ontanón, S., González-Calero, P. A. & Plaza, E. 2011. Measuring similarity in description logics using refinement operators. In *International Conference on Case-Based Reasoning*. Springer, 289–303.

Sazonau, V. & Sattler, U. 2017. Mining hypotheses from data in OWL: Advanced evaluation and complete construction. In *The Semantic Web - ISWC 2017 – 16th International Semantic Web Conference, Vienna, Austria, October 21–25, 2017, Proceedings, Part I*, **10587**. *LNCS*. Springer, 577–593.

Sazonau, V., Sattler, U. & Brown, G. 2015. General terminology induction in OWL. In *The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part I*, **9366**. *LCNS*. Springer, 533–550.

Shamsfard, M. & Barforush, A. A. 2003. The state of the art in ontology learning: A framework for comparison. *The Knowledge Engineering Review* **18**(4), 293–316. doi: 10.1017/S0269888903000687.

Sommer, E. 1995. An approach to quantifying the quality of induced theories. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) 95-Workshop on Machine Learning and Comprehensibility*. Citeseer.

Tran, A. C., Dietrich, J., Guesgen, H. W. & Marsland, S. 2017. Parallel symmetric class expression learning. *Journal of Machine Learning Research* **18**, 64:1–64:34.

Völker, J., Fleischhacker, D. & Stuckenschmidt, H. 2015. Automatic acquisition of class disjointness. *Journal of Web Semantics* **35**, 124–139.

Wogulis, J. L. 1994. *An Approach to Repairing and Evaluating First-Order Theories Containing Multiple Concepts and Negation*. PhD thesis, UCI.

Zemlyachenko, V., Korneenko, N. & Tyshkevich, R. 1985. Graph isomorphism problem. *Journal of Soviet Mathematics* **29**(4), 1426–1481. ISSN 0090-4104.