Syntactic Compression of Description Logics Terminologies

Raphael Melo
and Kate Revoredo
Postgraduate Information Systems Program
UNIRIO
Rio de Janeiro, Brazil
raphael.thiago, katerevoredo@uniriotec.br

Aline Paes
Department of Computer Science
Institute of Computing
UFF
Rio de Janeiro, Brazil
alinepaes@ic.uff.br

***Abstract*—**

Description Logics based languages have emerged as the standard knowledge representation scheme for ontologies. Typically, an ontology formalizes a number of dependent and related concepts in a domain, encompassed as a terminology. Usually, such terminologies are manually defined, which may yield unnecessarily large and complex terminologies, with a number of redundant group of literals among concepts. Simplicity has been always seen as a hallmark of good models, as they guide to faster inference and easier maintenance, compared to complex models. However, it is a hard task to manually convert a large and complex terminology, sometimes with about a hundred concepts definitions, to a simpler one. In this paper, we present an automatic refinement method to compress terminologies. The method relies on ideas of subtree isomorphism to convert a terminology into a simpler one without changing its semantic. Experimental results show that the proposed method successfully returns more compact terminologies while still maintaining the same set of individuals associated to a concept.

I. Introduction

Domain knowledge representation [4] is an important task when considering reasoning applications over a domain. Description logics (DLs) [1] form a family of representation languages, with different expressive power, 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 expressibility power, they have been widely used in Web Semantic [2] for Ontology representation.

The task of representing a domain knowledge through a DL is usually done manually, which is time consuming and may lead to overly complex ontologies, with several subsets of literals repeating in more than one concept definition. Such large ontologies usually takes much time to answer questions through inference and may also become unmanageable to maintain them, when the domain faces modifications. On the other hand, it is also a very difficult task to manually simplify a complex ontology, by choosing among sometimes hundreds of definitions the one(s) to be replaced by a simpler version, while still keeping the same semantic as before.

Simplicity has always been regarded as a hallmark of scientific descriptions. Occam's razor, or *Lex Parsimoniae*, is a logical principle that has been historically used within scientific discourse as a way to choose between competing hypotheses. The idea is well expressed using a quote from Aristotle's *Posterior Analytics*:

> We may assume the superiority *ceteris paribus* (all things being equal) of the demonstration which derives from fewer postulates or hypotheses.

Thus, in order to obtain compact terminologies, without letting this burden to the user of the domain, in this paper we present a method that automatically simplifies terminologies by identifying groups of repeated literals among concepts definitions. When a group of literals within a concept definition is logically equivalent to another concept's definition, the substituition of the group in the former by the latter concept name maintains the semantics while shortens the definition. We rely on the concept of isomorphism in trees to ensure this logical equivalency.

The paper is organized as follows. In Section II the background knowledge for descriptions logics are reviewed. In Section III, we present the proposed method for *syntactic compression* of terminologies. Section IV presents the experiments made to validate the proposal. Section V presents the related works. In Section VI we conclude the paper.

II. Description Logics

The family of knowledge representation languages largely used to formalize ontologies are called Description Logics [1]. They are decidable fragments of First-order logic which try to achieve a balance between expressibility and complexity.

A DL knowledge base ($\mathcal{KB}$) has two components: a *TBox* and a *ABox*. The *TBox* contains intensional knowledge in the form of a terminology, while the ABox is composed of assertions about the individuals of the domain. Knowledge is expressed in terms of *individuals*, *concepts*, and *roles*. Thus, the terminology consists of concepts, which denote a set of individuals and roles which denote binary relationships between individuals. This knowledge is represented in a formally defined syntax and semantics. Example 1 shows how a definition in natural language can be translated into a formally defined DL syntax.

Example 1. *Let the natural language definition for the "Father" concept be:"Father is a male parent"; one can define its DL syntax as:* $\mathsf{Father} \equiv male \sqcap \exists Parent.\top$

The semantic of a description is given by a *domain* $\mathcal{D}$ (a set) and an *interpretation* $\cdot^{\mathcal{I}}$ (a functor), where individuals represent objects through names from a set $N_{\mathsf{I}} = \{a, b, \ldots\}$, each *concept* in the set $N_{\mathsf{C}} = \{C, D, \ldots\}$ is interpreted as a subset of $\mathcal{D}$ and each *role* in the set $N_{\mathsf{R}} = \{r, s, \ldots\}$ is interpreted as a binary relation on $\mathcal{D}$. From here on, $a, b, a_1, b_1, \ldots$ are individuals; $C, D, C_1, D_1, \ldots$ are concepts; $r, r_1, r_2, \ldots$ are roles. The syntax is DL language dependent, thus, assuming a vocabulary $\mathcal{S}$, containing *individuals*, *concepts* and *roles* a DL language is defined by its set of constructors (language specific), which are used to combine the vocabulary elements. For instance, the DL $\mathcal{ALC}$'s syntax [16] is defined by the constructors (i) *intersection* ($C \sqcap D$), (ii) *union* ($C \sqcup D$), (iii) *complement* ($\neg C$), (iv) *existential restriction* ($\exists r.C$) and (v) *value restriction* ($\forall r.C$).

Concept *inclusions/definitions* are represented respectively by $C \sqsubseteq D$ and $C \equiv D$. In this paper, we assume the common assumption made about DL terminologies: (i) there is only one definition for a concept name and (ii) concept definitions are acyclic. The *ABox* is composed of assertions about the individuals of the domain. An assertion states that an individual belongs to a concept or that a pair of individuals satisfies a role. Attached to a description logic there must be a reasoning mechanism, responsible for inferring information about individuals from ($\mathcal{KB}$). Figure 1 depicts a $\mathcal{KB}$.

KB

TBox	ABox
Father ≡ Male⊓∃Parent.⊤	Male (ALFRED)
Mother≡Female⊓∃Parent.⊤	Male (CARL)
Wife≡Female⊓∃Married.Male	Female(BEATRICE)
Husband≡Male⊓	Female(CORNELIA)
∃Married.Female	Parent(BEATRICE, CORNELIA)
	Parent(ALFRED, CORNELIA)
	Father (ALFRED)
	Mother(BEATRICE)

Fig. 1. Example of a Knowledge Base

III. Syntactical Compression

The technique to compact existing definitions proposed in this paper is motivated by the fact that if part of the definition has a semantically equivalent but syntactically smaller counterpart, then a substitution could take place generating a smaller though equivalent definition. Henceforth we will call this process *syntactic compression*. In the following section, we present our method for *syntactic compression*.

A. Syntactic Compression using Tree Representation

In order to achieve a syntactic compression it is necessary to find viable substitutions, i.e., parts of a concept definition that encompass another concept. Example 2 illustrate a possible syntactic compression.

Example 2. *Let A, B, C, D, E concepts in a DL such that concept $A \equiv C \sqcap D \sqcup E$ and concept $B \equiv D \sqcup E$. The section $D \sqcup E$ within A encompasses concept B. Therefore, concept A can be rewrite as $A \equiv C \sqcap B$.*

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 DL constructors are commutable, as shown in Example 3.

Example 3. *The logical definition $B \sqcap C \sqcap D$ is equivalent with any conjunction of permutations of the concepts $\{B, C, D\}$, such as: (i) $B \sqcap D \sqcap C$, (ii) $D \sqcap B \sqcap C$, ...*

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. In this tree: (i) internal nodes represent a constructor, (ii) leaves represent concepts, and (iii) all children of a constructor node are commutable. Allowing the commutability 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 4.

Example 4. *Let A, B, C, D, E concepts in a DL such that concept $\mathsf{A} \equiv C \sqcap D \sqcap E$ and $\mathsf{B} \equiv E \sqcap C \sqcap D$. The tree representation for both concepts should be the following:*

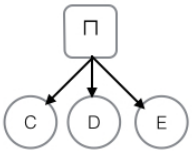


Figure 2 shows all the correlations between a possible constructor in $\mathcal{ALC}$ and its tree representation. We stress that the proposed method is language independent as long as it is possible to separate the constructors of the DL language between unary and binary (commutable n-ary) constructors.

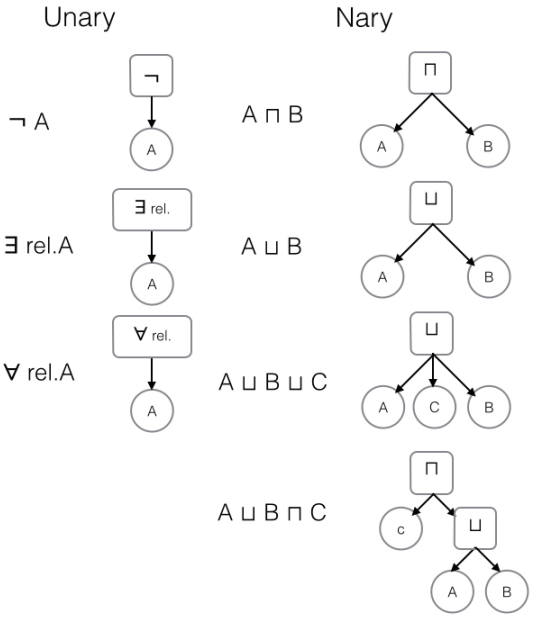


Fig. 2. $\mathcal{ALC}$ constructors to tree

B. Redundancy Removal on DLs Descriptions (R2D2)

Algorithm 1 presents our procedure to execute the syntactic compression of a terminology.

The algorithm receives as input a set of concepts descriptions (terminology), and a set of available constructors, divided into unary and n-ary, and returns a set of compressed concept descriptions.

Algorithm 1 Syntatical Compression Top-level Algorithm

Input: a set of concept definitions $C_d = \{C_1, \ldots, C_n\}$ and the set of valid constructors in the DL
Output: a set of compressed concepts definitions $C'_d = \{C'_1, \ldots, C'_n\}$

1: sort C_d by concept definition length;
2: $\mathcal{T}_c \leftarrow \{\}$
3: **for** each $C_i \in C_d$ **do**
4: $\mathcal{T}_c.add$ Algorithm 2(C_i, 0, C_i length, valid unary constructors, valid n-ary constructors); {Find the syntatic tree for C_i}
5: **for** each pair $\mathcal{T}_{ci}$ and $\mathcal{T}_{cj}$ in $\mathcal{T}_c$, where $i < j$ **do**
6: Algorithm 4($\mathcal{T}_{ci}$, $\mathcal{T}_{cj}$); {Syntatic compress $\mathcal{T}_{ci}$ with $\mathcal{T}_{cj}$}
7: **for** each $\mathcal{T}_{ci} \in \mathcal{T}_c$ **do**
8: $C_i \leftarrow$ TreeToDefinition($\mathcal{T}_{ci}$);
9: **return** C_d

It starts by sorting the set of concepts in ascending order of length (Definition 1). It is an important step, because a concept may only have a substitution involving a smaller concept.

Definition 1. *Length is the sum of all constructors and concepts presented in a concept definition, e.g., $A \equiv B \sqcap C \sqcap D$ has a length of 5.*

Next, for each concept in the terminology it finds its corresponding tree form (line 4). This step is better visualized through Algorithms 2 and 3, that we discuss later. Once it has the tree representation of 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 bigger one (line 5), and calling Algorithm 4 (line 6)to find a proper substitution. After all the substitutions are made, it changes 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.

a) Syntactic Tree Construction: Algorithms 2 and 3 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 19 of algorithm 2 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, a block may contain other blocks. The upper and lower bound are, respectively the begin and end of a concept or block.

Algorithm 2 Logical Description to Syntatical Tree

Input: a concept description C; a lower bound; an upper bound; a set of unary constructors $S_u = U_C1, \ldots, U_Cn$ and a set of nary constructors $S_n = N_C1, \ldots, N_Cn$
Output: A syntatical Tree $root$

1: create an empty $root$;
2: create the auxilary variable $firstConcept$;
3: $index \leftarrow lowerBound$;
4: **while** $index \leq upperBound$ **do**
5: $currentConstruct \leftarrow C[index]$;
6: **if** $currentConstruct$ is a block **then**
7: $aux \leftarrow$Algorihtm 2(C, index, end block index, S_u, S_n);
8: **if** $root$ is empty **then**
9: $root \leftarrow result$;
10: **else**
11: merge $root$ with $result$;
12: increment index;
13: continue to next loop iteration;
14: **if** $currentConstruct \in S_n$ **then**
15: **if** $root$ is empty **then**
16: $root \leftarrow currentConstruct$;
17: **if** $firstConcept$ is not empty **then**
18: add $firstConcept$ as a child of $currentConstruct$;
19: **if** $root \neq currentConstruct$ **then**
20: add $root$ as a child of $currentConstruct$;
21: $root \leftarrow currentConstruct$;
22: increment index;
23: continue to next loop iteration;
24: **if** $currentConstruct \in S_u$ **then**
25: $currentConstruct \leftarrow$ Algorithm 3;
26: $index \leftarrow$ end index returned in line 25;
27: **if** $root$ is empty **then**
28: $firstConcept \leftarrow currentConstruct$;
29: **else**
30: add $currentConstruct$ as a child of $root$;
31: **if** $root$ is empty AND $firstConcept$ is not empty **then**
32: $root \leftarrow firstConcept$;
33: **return** $root$

b) Finding Proper Substitutions: Algorithm 4 is used to perform substitutions between two trees. It receives two trees: a bigger tree and a smaller tree. To perform a substitution, it must find a node within the bigger tree that is the root of the smaller tree (line 4). An equivalent tree is found when: i) all the internal nodes in the smaller tree have an equivalent node in the bigger tree subtree (also in the right "position"), and ii) the subtree root of the bigger tree contains all the nodes below the smaller tree root. Example 5 shows why the second restriction does not state that the bigger 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*.

Example 5. *Let $F \equiv A \sqcup B \sqcap C$ and $E \equiv A \sqcup B$. Executing the substitution on the tree representation of F and E we have:*

Algorithm 3 Recursive Method for Unary Constructors

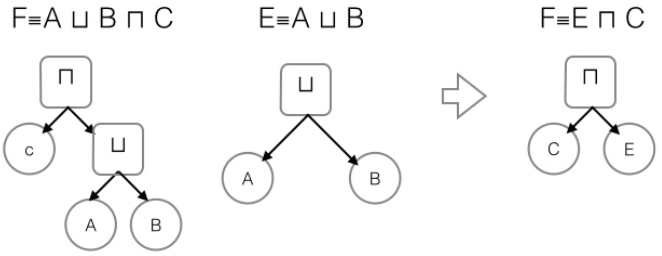
Input: a concept description C; a lower bound; a set of unary constructors $S_u = U_C1, \ldots, U_Cn$ and a set of nary constructors $S_n = N_C1, \ldots, N_Cn$
Output: A syntatical Tree $node$ and the ending index

```
index ← lowerbound;
currentConstruct ← C[index];
if currentConstruct ∈ S_u then
    if currentConstruct is a unary quantifiers then
        currentConstruct ← all items from index to relation statement;
    currentConstruct.descendent ← Algorithm 3;
    index ← Algorithm 3 index;
else
    if currentConstruct is a block then
        currentConstruct ← Algorithm 2(C, index, end block index, S_u, S_n);
        index ← end block's index;
return currentConstruct and index
```

Algorithm 4 Syntatical Compression's Substitution

Input: larger tree $\mathcal{T}_l$; smaller tree $\mathcal{T}_s$

```
nodesToVerify ← {T_l.root};
while nodesToVerify is not empty do
    rootLarger ← nodesToVerify.pop;
    if rootLarger ≡ T_s.root then
        rootLarger ← T_s concept;
    add rootLarger in nodesToVerify;
```



IV. Experimental Evaluation

In this section, we describe the experiments conducted towards evaluating the proposed *syntactic compression* method.

In order to validate the method we separated the experiments in two sets: a toy example on the kinship domain and a collection of 46 datasets to emulate real scenarios.

A. Toy Example

To verify the correctness of the proposed method, we used a small terminology in the kinship domain [12]. The terminology was composed of five concepts { *male, female, Grand Parent (GP), Grand Father(GF), Grand Mother(GM)*}, obeying their regular semantic; *male* and *female* are atomic concepts (concepts without terminological definition). The following is the TBox for this terminology:

- $GP \equiv \exists hasChild.\exists hasChild.\top$
- $GF \equiv male \sqcap \exists hasChild.\exists hasChild.\top$
- $GM \equiv female \sqcap \exists hasChild.\exists hasChild.\top$

We ran the implemented method over this terminology and it returned the following terminology:

- $GP \equiv \exists hasChild.\exists hasChild.\top$
- $GF \equiv male \sqcap GP$
- $GM \equiv female \sqcap GP$

As expected, the method returned a terminology where every possible substitution (GP into GF and GM) is made, achieving a more compact terminology. The first terminology has length 19, while the post method terminology has length 11. This shows the efficacy of the method.

B. Real world problems

In order to evaluate the beheaviour of the method in real scenarios we used a number of datasets from OAEI's 2013 campaign [1] of ontology matching and all datasets packaged as examples in DL-Learner system [2]. In total, there were 46 datasets with a wide range of domains, for example the kinship, moral, biomedical, ...and sizes, ranging from 0 to 10480 complex concepts.

Table I presents the results obtained from R2D2 with each one of the 46 datasets. Most of the datasets only have ABoxes, in theses cases the method does not proposes any replacements on the original terminology since there is nothing to compress. In most of the other cases the method had no impact on the size of the terminology. With further analysis this happenend because the terminologies were already syntactically compressed, then the terminology after the method has the same length as earlier. More than anything, this might be a testament of the quality of the terminologies.

The method achieved *syntactic compression* on only three of the forty-six terminologies: i)conf_Cocus; ii)oaei2013_NCI_small_overlapping_fma.owl; iii)oaei2013_NCI_whole_ontology.owl. Next we analyse each substitution:

- *conf_Cocus*: the original terminology had only 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 definition by the other concept, as an example resulting in: $Event \equiv Document$. In this particular case it might be argued that the real meaning is lost when this substituion is performed, but from a pure logical point of view the semantics is mantained.
- *oaei2013_NCI_small_overlapping_fma.owl* and *oaei2013_NCI_whole_ontology.owl*: both terminologies pertain to the same domain of biomedicine, sharing much of the concepts and definitions.

[1] http://oaei.ontologymatching.org/
[2] http://dl-learner.org/Projects/DLLearner

TABLE I. RESULTING METHOD

Dataset	#Concepts	Original Size	Post Size	Exec Time in secs.
DL—Arch	5	18	18	.018273
DL—Family-Father	1	2	2	.001455
DL—Family-Uncle	1	3	3	.004273
DL—Forte-Family	1	3	3	.001455
DL—MoralReasoner	20	133	133	.032727
DL—MoralReasoner(43)	19	130	130	.034091
DL—MoralReasoner(43 Complex)	17	112	112	.025364
OAEI—conf_cmt	1	5	5	.001364
OAEI—conf_Cocus	5	19	**17**	.005727
OAEI—conf_Conference	13	49	49	.024364
OAEI—conf_confious	8	24	24	.011273
OAEI—conf_edas	7	42	42	.011091
OAEI—conf_PCS	4	12	12	0
OAEI—conf_sigkdd	7	21	21	.01
OAEI—NCI_small_overlapping_fma	1964	17046	17046	55.59373
OAEI—SNOMED_small_overlapping_fma	2882	35552	35552	86.63409
OAEI—NCI_small_overlapping_snomed	6851	74191	**74183**	684.26991
OAEI—NCI_while_ontology	10280	151316	**151308**	1997.30955
Rest of datasets (#28)	0	0	0	0

Looking closely we can see that the terminology oaei2013_NCI_small_overlapping_fma.owl is a subset of oaei2013_NCI_whole_ontology.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 had a structure of many conjunctions of existential restrictions. These particular replacements are difficult to visualize because of the size of the concepts (41 and 57 respectively). This might suggest a direct relationship between a terminology size and complexity and how difficult it is to find all simplifications. For humans there seems to be a limit over which a terminology or concept is too complex to handle all possible substitutions.

From these experiments it appears that the use of the method, provided that the computational cost are not too high, has the potential to yield terminologies as simple as syntactically possible.

C. R2D2 Computational Behaviour

In this section we try to establish the relationship between the execution time of the R2D2 and the amount of concepts and size of a terminology.

To evaluate this we ran the experiment of section IV-B ten times and calculate the average execution time for each terminology. Figure 3 shows the relationship between the number of concepts/size of terminology and the execution time.

Although, it seems that, at some point, the execution time increases exponentially as terminology gets bigger, it is highly unlikely that terminologies would have much more than 10.280 concepts and 151.316 in size. Having 10280/151316 as a soft limit for complexity, 20 to 30 minutes execution time seems reasonable.

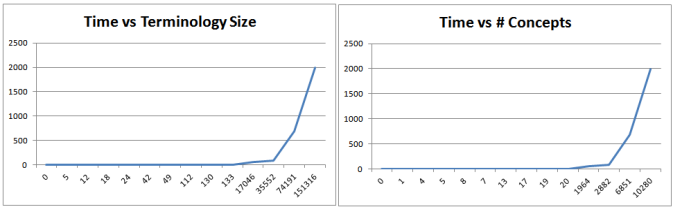


Fig. 3. Execution time vs Number of concepts and Terminology size

V. RELATED WORKS

The syntactic compression method that we developed here automatically restructures a previously defined terminology. [19] presents a survey of works on theory refinement for First Order Logic. *Theory refinement* can be divided into *theory revision* and *theory restructuring*. Theory revision is the change of an existing theory to correctly represent a change in the domain, e.g., when new examples are wrongly classified by the theory. Theory restructuring is the process of adjusting an existing theory to match an external criteria such as understandability or efficiency, differing from revision because the theory semantics remains unchanged. Most of the work on theory refinement focus either on revision or restructuring for the purpose of efficiency. Wiht respect to such a context, this paper is classified as theory restructuring for understandability (simplicity), having a collaterall effect on efficiency.

Within this topic, FENDER [18], which is a method of *stratification* of theories, i.e., it structures a theory into layers of dependencies, has the same goal. It increases the number of layers, i.e., the dependencies within a theory. To do so, it uses an operator named *common partial premisses*, which finds common groups of literals within the theory and creates new predicates from them thus increasing the dependencies while reducing redundancy. FENDER and R2D2 are similar in the sense that both have the same goal and a similar method. They differ because R2D2 does not create new concepts (predicates), i.e., the common groups can only be equal to an existing

concept definition. Another theory restructuring system is PFORTE_PI [14], [15], which uses the notion of predicate invention to restructure a probabilistic first-order theory, thus differing from R2D2 in the way the compression is made.

Additionally, there are two sets of works related to ours: i) tree representation of logical definitions, FOL [8] and DLs [6], and ii) tree isomorphism. In the former, their representation differs from ours in form, as their objective differs from ours. In general they use the tree representation to improve the scalability of reasoning. To that extend the structure is, at large, and ordered binary tree (in some cases the trees can have a larger degree). Because of the commutability of n-ary constructors we can not use ordered trees as a representation, as it would assume an order among the children, which is not suitable for syntactic compression.

Algorithm 4 presented in section III-A finds equivalent subtrees within a tree according to a matching tree. The equivalence of trees is well-known in the literature as *graph isomorphism* or more specifically *tree isomorphism*. Graph isomorphism is a NP-Complete problem [13], however in special cases it has polynomial solutions: trees are in this group of special cases [3], [5], [7], [11], [17]. We leave the investigation of more efficient tree subgraph algorithms for future works.

VI. Conclusions

In this paper, we proposed a method for syntactic compression of DL terminologies. The semantics of a concept is maintained, while its sintaxe is simplified, by performing substitutions among concepts. These substitutions are found using a tree representation of the concept's logical description. We developed algorithms to translate and to make the syntactical substitutions.

We validated our method's efficacy in two sets of problems: a toy example of the kinship domain and in a set of 46 datasets. Among the datasets were some particularly large terminologies, one of them with over ten thousand concepts. We then went on to verify the relationship between the execution time and a terminology complexity. Analyzing the results, we concluded that the relationship is exponential but, as it seems, only prohibiting on very large terminologies, in the tens of thousands range; yet the method had reasonable execution times in the low tens of thousand: around 30 minutes for a terminology with 10280 concepts. In terminologies with less than two thousand concepts we achieved runtime below the one minute mark. From the results, we came to the conclusion that using the proposed method is advisable with a upward soft limit of 10280 concepts.

In future work, we would like to investigate if there is any gain in using this method in a terminology learning process [9], [10] and also to investigate alternative more efficient tree isomorphism algorithms.

Acknowledgments

The first author would like to thank CAPES for the financial support through a master scholarship. This work is partially supported by FAPERJ through grant E-26/110.477/2014.

References

[1] F. Baader and W. Nutt. Basic description logics. In Franz Baader, Debora L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The description logic handbook*, pages 47–100. Cambridge University Press, 2 edition, may 2010.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 5(284):34, 2001.

[3] Hans L Bodlaender. Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631 – 643, 1990.

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

[5] SamuelR. Buss. Alogtime algorithms for tree isomorphism, comparison, and canonization. In Georg Gottlob, Alexander Leitsch, and Daniele Mundici, editors, *Computational Logic and Proof Theory*, volume 1289 of *Lecture Notes in Computer Science*, pages 18–33. Springer Berlin Heidelberg, 1997.

[6] Diego Calvanese. Finite model reasoning in description logics. *KR*, 96:292–303, 1996.

[7] Jean-Loup Faulon. Isomorphism, automorphism partitioning, and canonical labeling can be solved in polynomial-time for molecular graphs. *Journal of Chemical Information and Computer Sciences*, 38(3):432–444, 1998.

[8] Nuno Fonseca, Ricardo Rocha, Rui Camacho, and Fernando Silva. Efficient data structures for inductive logic programming. In *Inductive Logic Programming*, pages 130–145. Springer, 2003.

[9] Raphael Melo, Kate Revoredo, and Aline Paes. Learning multiple description logics concepts. In *ILP 2013 Late Breaking Papers "to appear"*, 2013.

[10] Raphael Melo, Kate Revoredo, and Aline Paes. Terminology learning through taxonomy discovery. In *2013 Brazilian Conference on Intelligent Systems*, pages 169–174, 2013.

[11] B.T. Messmer and H. Bunke. Subgraph isomorphism detection in polynomial time on preprocessed model graphs. In StanZ. Li, DineshP. Mital, EamKhwang Teoh, and Han Wang, editors, *Recent Developments in Computer Vision*, volume 1035 of *Lecture Notes in Computer Science*, pages 373–382. Springer Berlin Heidelberg, 1996.

[12] Lewis Henry Morgan. *Systems of consanguinity and affinity of the human family*. Smithsonian Institution,, 1870.

[13] Ronald C. Read and Derek G. Corneil. The graph isomorphism disease. *Journal of Graph Theory*, 1(4):339–363, 1977.

[14] Kate Revoredo, Aline Paes, Gerson Zaverucha, and Vítor Santos Costa. Combining predicate invention and revision of probabilistic fol theories. In *Short paper proceedings of 16th International Conference on Inductive Logic Programming (ILP-06)*, pages 176–178, 2006.

[15] Kate Revoredo, Aline Paes, Gerson Zaverucha, and Vítor Santos Costa. Combinando inveno de predicados e reviso de teorias de primeira-ordem probabilsticas. In *VI ENIA*, 2007.

[16] Manfred Schmidt-Schauss and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26, 1991.

[17] R. Shamir and D. Tsur. Faster subtree isomorphism. In *Theory of Computing and Systems, 1997., Proceedings of the Fifth Israeli Symposium on*, pages 126–131, Jun 1997.

[18] Edgar Sommer. Fender: An approach to theory restructuring. In *Machine Learning: ECML-95*, pages 356–359. Springer, 1995.

[19] Stefan Wrobel. First order theory refinement. *Advances in inductive logic programming*, 32:14–33, 1996.