



UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ON THE LEARNING OF MULTIPLE CONCEPTS IN
DESCRIPTION LOGIC

Raphael Melo Thiago

Orientadoras

Kate Cerqueira Revoredo
Aline Marins Paes Carvalho

RIO DE JANEIRO, RJ - BRASIL
MAIO DE 2014

ON THE LEARNING OF MULTIPLE CONCEPTS IN DESCRIPTION LOGIC

Raphael Melo Thiago

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA.

Aprovada por:

Kate Cerqueira Revoredo, D.Sc. - UNIRIO

Aline Marins Paes Carvalho, D.Sc. - UFF

Sean Wolfgang Matsui Siqueira, D.Sc. - UNIRIO

Fabio Gagliardi Cozman, Ph.D. - USP

RIO DE JANEIRO, RJ - BRASIL
MAIO DE 2014

Melo, Raphael Thiago
GXXXX On learning multiple concepts in Description Logics
/ Raphael Melo Thiago, 2014.
xv, 78f. ; 30 cm

Orientadora: Kate Cerqueira Revoredo.
Co-orientadora: Aline Marins Paes Carvalho
Dissertação (Mestrado em Informática) – Universidade Federal do
Estado do Rio de Janeiro, Rio de Janeiro, 2014.

1. XXXXX. 2. XXXXX. 3. XXXXX
4. XXXXXX. I. Revoredo, Kate Cerqueira
Vieira. II. Universidade Federal do Estado do Rio de Janeiro. Centro de
Ciências Exatas e Tecnológicas. Curso de Mestrado em Informática. III.
Título.

CDD - XXX

Resumo

Linguagens baseadas em Lógicas de Descrição são adotadas como o esquema padrão para a representação de conhecimento em ontologias. Tipicamente, uma ontologia formaliza um número de conceitos dependentes e relacionados de um domínio, agrupados como uma terminologia. Como a definição manual de tais terminologias é complexa, consome tempo e passível de erros, existe um grande interesse e demanda por métodos automáticos de aprendizado de terminologias. No entanto, as abordagens existentes seguem uma estratégia de aprendizado de um único conceito, desconsiderando dependências que possam existir entre conceitos. Como consequência, são induzidas terminologias mais complexas e por vezes ilegíveis. Logo, métodos para o aprendizado de vários conceitos dentro de uma tarefa, respeitando suas dependências são essenciais para a indução automática de ontologias compactas e compreensíveis. Assim, neste trabalho, propomos três estratégias para o aprendizado de terminologias compostas por múltiplos conceitos relacionados. Nós empiricamente avaliamos com sucesso todas as três em dois *benchmarks* e as comparamos com um algoritmo padrão de aprendizado de conceitos únicos.

Palavras-chave: Aprendizado de Múltiplos Conceitos em Lógicas de Descrição, Aprendizado de Terminologias, Remoção de Redundâncias em Lógicas de Descrição, Compressão de Terminologias, Restruturação de Teorias.

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. As defining such terminologies manually is a complex, time consuming and error-prone task, there is great interest and even demands for methods that learn terminologies automatically. However, the existing approaches follow a single concept learning strategy, disregarding dependencies that may exist among the concepts. As a consequence, a more complex and sometimes illegible terminology may be induced. Thus, methods for learning all the concepts within an unique task, respecting their dependency are essential for automatically inducing compact and understandable ontologies. Then, in this work, we propose three strategies for learning a terminology composed of multiple related concepts. We empirically evaluate successfully all of them in two benchmarks and compared them with a standard single concept learning algorithm.

Keywords: Multiple Concept Learning in Description Logics, Terminology Learning, Redundancy Removal on Description Logics, Terminology Compression, Theory Restructure.

LIST OF FIGURES

2.1	Architecture of a knowledge representation system based on Description Logics (Baader and Nutt, 2010).	8
2.2	Example of a DL Knowledge Base using the language \mathcal{AL}	10
3.1	The relationship among concepts C_i and C_j through their positive examples sets, represented as Venn diagrams.	23
3.2	An example of a taxonomy depicting the relation among concepts C_1, C_2, C_3, C_4 and C_5	24
3.3	Example 8's syntactic tree representation.	34
3.4	The syntactical tree representation of \mathcal{ALL} 's constructors.	35
4.1	R2D2's Execution time vs Number of concepts and Terminology size.	48
4.2	F-measure for Exp1 considering scenarios with BTL, DTL, ATL and DL-Learner (DL-L).	54
4.3	F-measure for Exp2 considering scenarios with BTL, DTL, ATL and DL-Learner (DL-L).	55
4.4	Size of found terminology for Exp1 and Exp2 with methods BTL, DTL, ATL and DL-Learner (DL-L).	56

4.5	Size of concepts definitions for Exp1 with methods BTL, DTL, ATL and DL-Learner (DL-L).	56
4.6	Size of concepts definitions for Exp2 with methods BTL, DTL, ATL and DL-Learner (DL-L).	57
4.7	Size difference for Exp1 w.r.t. Original Definition with methods BTL, DTL, ATL and DL-Learner (DL-L).	58
4.8	Size difference for Exp2 w.r.t. Original Definition with methods BTL, DTL, ATL and DL-Learner (DL-L).	58
4.9	Size difference for Exp1 w.r.t. SCL with methods BTL, DTL and ATL.	59
4.10	Size difference for Exp2 w.r.t. SCL with methods BTL, DTL and ATL.	59
5.1	Taxonomy of theory refinement tasks (Wrobel, 1996).	64

LIST OF TABLES

2.1	Syntax and Semantics for DL's Constructors.	9
3.1	Example of the output of a terminology learning task with and without the independence assumption.	18
3.2	Matrix of advantages and disadvantage for BTL, DTL and ATL. . . .	41
4.1	BTL: Best Threshold Experiment.	45
4.2	DTL's validation experiment results.	46
4.3	ATL: Results of the validation experiment.	46
4.4	Toy Example results: Concepts Definitions and their sizes found with the methods BTL, DTL, ATL and DL-Learner (DL-L).	51
4.5	Aggregate Results of Relaxing Completeness Experiment for the methods BTL, DTL and ATL.	53
4.6	Full Completeness over the individuals Experiment with the methods BTL, DTL and ATL.	54
4.7	Full Completeness Experiment ATL's Hits.	54

CONTENTS

1	Introduction	1
1.1	Goals	3
1.1.1	Before Terminology Learning	3
1.1.2	During Terminology Learning	3
1.1.3	After Terminology Learning	4
1.2	Organization	4
2	Background Knowledge	6
2.1	Description Logics	6
2.2	Inductive Logic Programming (ILP)	9
2.3	Concept Learning in Description Logics	12
2.4	Model's Simplicity and how it affects its understandability	15
3	Multiple Concept Learning	17
3.1	Before Terminology Learning (BTL)	21
3.1.1	Concepts Dependency	21
3.1.2	Learning Concept Ordering	24
3.1.3	Learning Terminology through concept ordering	26

3.2	During Terminology Learning (DTL)	28
3.2.1	DTL with cycle removal	31
3.3	After Terminology Learning (ATL)	32
3.3.1	Syntactic Compression using Tree Representation	33
3.3.2	Redundancy Removal on DLs Descriptions (R2D2)	35
3.4	Summary of methods	39
4	Experiments	42
4.1	Datasets	42
4.2	Experiments for each individual method	44
4.2.1	BTL: Finding the Best Threshold value	44
4.2.2	DTL	45
4.2.3	ATL	46
4.3	Experiments comparing methods	48
4.3.1	Experimental Methodology	49
4.3.2	Experimental Results	50
5	Related Works	61
5.1	Terminology Learning	61
5.2	Theory Restructuring - R2D2	63
6	Conclusions	66
6.1	Contributions	68
6.2	Future Works	69
A	OWA and CWA and their relationship with inferential and inductive reasoning	76

1 INTRODUCTION

The original intention for the World Wide Web was to provide a distributed collection of documents, connected via Hyperlinks accessible via the Internet. As is, the unstructured nature of the information on the Web, although easily accessible by humans, provide a challenge for machines to efficiently analyse it. Berners-Lee et al. (2001) proposes a road map to transform the web from a machine hostile environment into a place where machine reasoning can be “ubiquitous and devastatingly powerful”. This set of ideals are called as *Semantic Web*.

In order for the machines to be capable of analysing the information available on the Web, the information must be structured. *Ontologies* are formally defined knowledge bases, they provide the needed structure for the information on the Semantic Web.

There are two major challenges to Semantic Web’s usage: i) a standard knowledge representation formalism and ii) the lack of structured knowledge in the form of ontologies.

Knowledge representation (Brachman and Levesque, 2004) is an important task when considering applications that require reasoning over a domain. *OWL Web Ontology Language* (OWL) ¹ has become a W3C Recommendation in February 2004,

¹<http://www.w3.org/TR/owl-features/>

it is the current standard for defining ontologies to the Semantic Web. It comprises of three sublanguages with increasing expressiveness. The three sublanguages are: OWL Lite, OWL DL, and OWL Full. Its vocabulary can be directly related to the formal semantics of Description Logics (DLs) (Baader and Nutt, 2010).

DLs form a family of representational languages, with different expressive power, that are typically decidable fragments of first order logic (FOL) (Huth and Ryan, 2004). They represent domain concepts and relations between them.

The lack of readily available ontologies indicates the need for applications that support the creation and maintenance of them, using existing background knowledge. One way of resolving this challenge is to manually define DL ontologies, however this process is time consuming and error prone. Therefore, the consideration of mechanisms for automatically learning (Anderson et al., 1986) a model represented in DL is relevant and sometimes even demanding. A number of approaches have been proposed in the literature, such as the algorithms proposed in (Lehmann and Hitzler, 2010), (Fanizzi et al., 2008) and (Iannone et al., 2007a). Furthermore, usually, domains encompass more than one non-primitive concept. However, to the best of our knowledge, the description logic learning approaches devised so far follow a single learning strategy, i.e., focus on learning piecewise concepts. When learning multiple concepts, independent executions of the learning algorithm are performed for each one of the concepts, thus possible relations among them are ignored. We argue that when the several concepts belonging to a DL are learned jointly, the final terminologies may be more compact, and therefore more useful. Moreover, learning jointly the concepts may yield more accurate concepts than usual approaches, since each concept may have in its definition others concepts previously defined, as a starting point.

The main hypothesis we would like to investigate is:

“If dependencies among the concepts of the domain that need to be learned are found then clearer and compact terminologies are yielded while their quality is maintained.”

1.1 GOALS

To investigate the hypothesis we developed three auxiliary hypotheses related to the discovery of dependencies among the group of concepts that will be learned. These methods are the main contributions of this work. Next, we briefly introduce them.

1.1.1 BEFORE TERMINOLOGY LEARNING

Before Terminology Learning's (BTL) hypothesis is: if dependencies among the concepts that will be learned are found *before* the learning process then clearer and compact terminologies are yield while maintaining their quality.

BTL (Melo et al., 2013b) finds the dependencies before the concepts are learned. The dependencies are found comparing the examples used in the learning tasks. The intersection of the sets of examples indicates the relationship among the concepts. From this heuristic, a taxonomy that models the *subsumer/subsumee* dependencies is build. This taxonomy indicates a learning order. A concept learned in an iteration can use the concepts defined in previous iterations, this yields concept definitions that match the dependencies.

Because the taxonomy has a bias towards dependencies of the *subsumer/subsumee* type, if the best terminology contains another type of dependency, this terminology may not be found.

1.1.2 DURING TERMINOLOGY LEARNING

During Terminology Learning's (DTL) hypothesis is: if dependencies among the concepts that will be learned are found *during* the learning process then clearer and compact terminologies are yield while maintaining their quality.

The second method (Melo et al., 2013a) leaves to the concept learning method the task of finding the dependencies. The examples of other learning tasks are inserted into the background knowledge as facts for a learning task. Those facts act as surrogates for the later found concept definitions. This allows for a greater diversity in the types of dependencies found, w.r.t. those found by BTL, overcoming BTL’s bias for *subsumer/subsumee* dependencies.

1.1.3 AFTER TERMINOLOGY LEARNING

After Terminology Learning’s (ATL) hypothesis is: if dependencies among the concepts are found within a terminology then a clearer and compact version of the terminology can be found while maintaining their quality.

ATL (Melo et al., 2014) identifies dependencies by finding reused definition’s groups syntactically equivalent to full definitions within a defined terminology. The substitution of a group by its equivalent concept yields a smaller terminology while maintaining the same semantics. This method can be applied *after* a terminology is obtained, be it via a manual definition of a terminology learning task, hence its name.

An experimental evaluation was conducted comparing the three approaches and its variations. Moreover, we compared them to DL-Learner (Lehmann and Hitzler, 2010), a framework that performs single concept learning tasks, showing the benefits of a multiple concept learning algorithm over a single concept learning algorithm.

1.2 ORGANIZATION

The paper is organized as follows. Chapter 2 presents the background knowledge necessary to understand the proposed approaches. In Chapter 3, our three approaches for multiple concept learning are described in detail. Chapter 4 presents

the experiments conducted to validate our proposals. Chapter 5 presents the works related to this research. Finally, Chapter 6 presents the closing remarks of this research, a summary of the contributions and directions for future works.

2 BACKGROUND KNOWLEDGE

In this chapter we will present the background knowledge necessary to understand this work. First we will present the foundations of *Description Logics*, discussing its syntax and semantics. A firm grasp of DLs is necessary to understand the concept definitions found and the inner workings of the methods.

Secondly we will present the problem of learning a concept definition. First the ideas of *Inductive Logic Programming* will be shown, because it provides the bases for concept learning in Description Logics, defined just after.

Lastly, we will present formal arguments supporting the motivation for methods to simplify terminologies and the relationship between simplicity and understandability.

2.1 DESCRIPTION LOGICS

Description Logics (DL) (Baader and 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 (\mathcal{KB}). Description Logics contains a formal logic-based semantics, hence its name. DLs are also characterized by its focus on reasoning as a service: allows the inference of implicit knowledge from the explicit knowledge contained in the knowledge base (\mathcal{KB}). It has two reasoning services: i) classification of concepts and ii) classification of individuals. The classification of concepts determines subconcept-superconcept relationships between them (also called subsumption relationship in DL). The classification of individuals determines whether an individual is an instance of a certain concept.

Description Logics are useful for Semantic Web because, unlike *First Order Logic* (FOL) theories, they are decidable, i.e., a query is always answered, both positive and negatively. However, the *decidability* and *complexity* depends on the Description Logic language chosen. There is a tradeoff between expressiveness and the reasoning complexity. A higher expressive power incurs in a higher reasoning complexity, on the other hand, a lower expressive power incurs in a lower reasoning complexity.

A *Knowledge Base* (\mathcal{KB}) has two components: a *ABox* and a *TBox*. The TBox contains the *terminology*, i.e., the domain's vocabulary, while the *ABox* contains assertions about the individuals using the terms defined in the terminology. Fig. 2.1 displays the architecture of a knowledge representation system based on Description Logics.

Atomic concepts and *roles* are the elementary descriptions. Complex descriptions, or *descriptions* in short form, can be built from then 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, \dots for *individuals*; the letters A, B, A_1, B_1, \dots for *atomic concepts*, the letters R, R_1, \dots for *atomic roles* and the letters C, D, C_1, D_1, \dots for *concept descriptions*. An *assertion* has the form $C(a)$ (*concept assertion*) or $R(a, b)$ (*role assertion*),

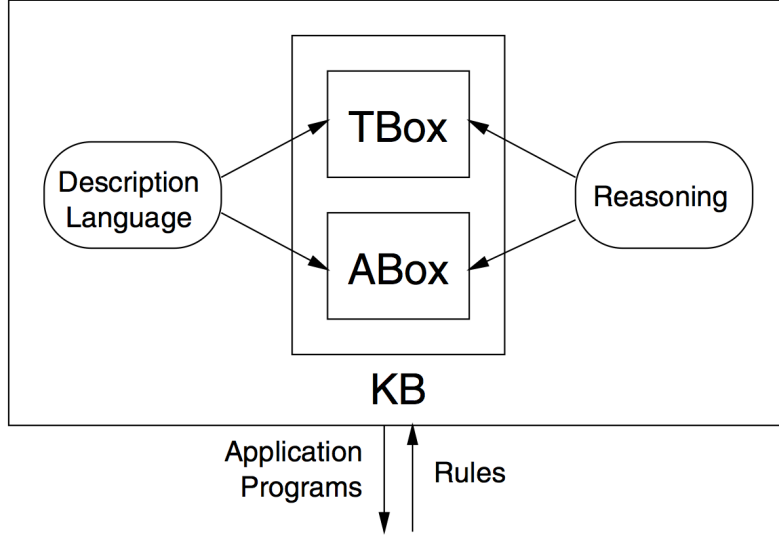


Figure 2.1: Architecture of a knowledge representation system based on Description Logics (Baader and Nutt, 2010).

Table 2.1 displays the syntax and semantics for various DL constructors. The first column specifies the constructor’s name. The second column specifies the letter that denote the constructor when naming the logic. The syntax of each constructor is given in the third column.

The language \mathcal{AL} (Schmidt-Schauss and Smolka, 1991) is the minimal DL language, its constructors are the set within Table 2.1 with no letter associated. The other DL languages are extensions of \mathcal{AL} . Example 1 shows how a definition in natural language can be translated into a concept description using the language \mathcal{AL} , while Fig. 2.2 shows an example of a complete \mathcal{KB} .

Example 1. *Let the natural language definition for the “Father” concept, in the kinship domain, be:*

“Father is a male parent”; one can define its concept description as:

$$\text{Father} \equiv \text{male} \sqcap \exists \text{Parent}.\top$$

The forth column of Table 2.1 displays the semantics for the constructors. The semantics of a description is given by a *domain* \mathcal{D} (a set) and an *interpretation* \mathcal{I} (a functor). Individuals represent individuals through names from a set $N_I = \{a, b, \dots\}$. Each *concept* in the set $N_C = \{A, B, C, D, A_1, B_1, \dots\}$ is interpreted as a subset of a

Table 2.1: Syntax and Semantics for DL's Constructors.

Constructor's Name		Syntax	Semantics
atomic concept		A	$A^{\mathcal{I}} \subseteq \mathcal{D}^{\mathcal{I}}$
universal concept		\top	$\mathcal{D}^{\mathcal{I}}$
bottom concept		\perp	\emptyset
atomic negation		$\neg A$	$\mathcal{D}^{\mathcal{I}} / A^{\mathcal{I}}$
intersection		$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
value restriction		$\forall R.C$	$\{o \forall o' : (o, o') \in R^{\mathcal{I}} \rightarrow o' \in C^{\mathcal{I}}\}$
limited existential quantification		$\exists R$	$\{o \exists o' : (o, o') \in R^{\mathcal{I}}\}$
existential quantification	\mathcal{E}	$\exists R.C$	$\{o \exists o' : (o, o') \in R^{\mathcal{I}} \wedge o' \in C^{\mathcal{I}}\}$
disjunction	\mathcal{U}	$C \sqcup D$	$C_1^{\mathcal{I}} \cup D_2^{\mathcal{I}}$
general negation	\mathcal{C}	$\neg C$	$\mathcal{D}^{\mathcal{I}} \setminus C^{\mathcal{I}}$
qualified number restrictions	\mathcal{Q}	$\exists^{\geq m} R$	$\{o \#\{o' (o, o') \in R^{\mathcal{I}}\} \geq m\}$
		$\exists^{\leq n} R$	$\{o \#\{o' (o, o') \in R^{\mathcal{I}}\} \leq n\}$
well-founded	\mathcal{W}	$wf(R)$	$\{o_0 \forall o_1, o_2, \dots (\text{ad infinitum}) \exists i \geq 0 : (o_i, o_{i+1}) \notin R^{\mathcal{I}}\}$
role value map	\mathcal{V}	$R_1 \subseteq R_2$	$\{o \{o' (o, o') \in R_1^{\mathcal{I}}\} \subseteq \{o' (o, o') \in R_2^{\mathcal{I}}\}\}$
role name		R	$R^{\mathcal{I}} \subseteq \mathcal{D}^{\mathcal{I}} \times \mathcal{D}^{\mathcal{I}}$
inverse	\mathcal{I}	R^-	$\{(o, o') (o', o) \in R^{\mathcal{I}}\}$
union	\mathcal{R}	$R_1 \cup R_2$	$R_1^{\mathcal{I}} \cup R_2^{\mathcal{I}}$
concatenation	\mathcal{R}	$R_1 \circ R_2$	$R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}}$
reflexive transitive closure	\mathcal{R}	R^*	$(R^{\mathcal{I}})^*$
identity	\mathcal{R}	$id(C)$	$\{(o, o) o \in C^{\mathcal{I}}\}$
difference	\mathcal{D}	$R_1 \setminus R_2$	$R_1^{\mathcal{I}} \setminus R_2^{\mathcal{I}}$

domain \mathcal{D} . Each *role* in the set $N_R = \{R, R_1, \dots\}$ is interpreted as a binary relation on the domain.

An interpretation \mathcal{I} satisfies an *inclusion axiom* $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it satisfies an *equality axiom* $C \equiv E$ if $C^{\mathcal{I}} = E^{\mathcal{I}}$. An interpretation that satisfies all axioms in a terminology \mathcal{T} is called a *model* of \mathcal{T} . A *finite model* is a model with finite domain. A concept description C is (finitely) *consistent* in \mathcal{T} if \mathcal{T} admits a (finite) model \mathcal{I} such that $C^{\mathcal{I}} \neq \emptyset$ and C is (finitely) *subsumed* by D in \mathcal{T} if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every (finite) model \mathcal{I} of \mathcal{T} .

Moreover, 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.

2.2 INDUCTIVE LOGIC PROGRAMMING (ILP)

Modeling the knowledge of a domain is an essential task in computer and information science. However, manually formalizing the domain knowledge is an expensive and error prone task, even more because the domain experts themselves do not always agree about definitions and their relationships-(Maedche and Staab, 2001).

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

Figure 2.2: Example of a DL Knowledge Base using the language \mathcal{AL} .

Therefore, in order to effectively exploit such terminologies it is necessary to apply techniques from machine learning (Mitchell, 1997) for automatically inducing terminologies from data.

Inductive Logic Programming (ILP) is a research field in the intersection of machine learning and logic programming (Lavrac and Dzeroski, 1994; Muggleton and De Raedt, 1994). ILP research has the goal of finding learning algorithms for inducing logic programs that represents relational definitions.

Logic Programming is a programming paradigm based on formal logic. It uses a strict form of First-Order Logic (FOL) theories, where *sentences* can only be *clauses* (disjunction of *literals*¹) with at most one positive literal. These types of clauses are know as *Horn Clauses* (Horn, 1951).

A Horn Clause with exactly one positive literal is a called a *definite clause*. A definite clause with no negative literal is called a *fact*. A Horn Clause with no positive literal is called a *goal clause*.

Formally defined logical theories are useful because of the ability to infer new implicit knowledge that follows from the explicit background knowledge \mathcal{BK} (*inferential reasoning*). The rules of inference defines that given a sentence (or a set of sentences) with certain properties a sentence (or a set of sentences) can be derived

¹A literal is a predicate or the negation of a predicate.

as a conclusion. A rule is *sound* when it preserves the truth-value for any interpretation, a rule is also *complete* if every logically valid sentence is derivable. The inference of new information is done through a set of sound *inference rules*.

The *Resolution rule* was presented by Robinson (1965). Resolution is a valid inference rule (sound and complete), that can, without any other inference rule, be used to build a sound and complete theorem proving method. Iteratively applying the resolution rule in a proper way allows for finding if a FOL clause is satisfiable.

Horn Clauses are important because their properties² leads to an efficient theorem proving method (Dowling and Gallier, 1984).

ILP methods has as goal to find a hypothesis \mathcal{H} that best defines a given set of positive and negative examples taking into account a background knowledge (\mathcal{BK}) (Definition 1). The hypothesis, \mathcal{BK} and examples are logical programs.

Definition 1 (Concept Learning in ILP (Muggleton and De Raedt, 1994)). *Given:*

- a base theory \mathcal{BK} (a logical program with Horn clauses);
- a set of positive examples \mathcal{E}^+ ;
- a set of negative examples \mathcal{E}^- ;

Find a logical program \mathcal{H} on which the following conditions hold:

- every positive example $e \in \mathcal{E}^+$ is covered by $\mathcal{H} \wedge \mathcal{BK}$ (complete);
- no negative example $e \in \mathcal{E}^-$ is covered by $\mathcal{H} \wedge \mathcal{BK}$ (consistent).

An example is covered by a logical program \mathcal{H} if it can be logically inferred from it.

The problem of solving a ILP learning task can be regarded as a search problem, where the space of solutions are: (i) the “well formed” set of hypotheses and (ii) a acceptance criterion for the candidate hypotheses³.

²The resolvent of two Horn Clauses is itself a Horn Clause; the resolvent of a goal clause and a definite clause is a goal clause.

³Along with coverage, parsimony is also a goal, i.e., smaller theories are preferable.

2.3 CONCEPT LEARNING IN DESCRIPTION LOGICS

Since concepts are the basic components of terminologies, the first attempt on machine learning would be how to learn a concept definition from available data, this is called *Concept Learning in Description Logics*, henceforth *Concept Learning* or SCL. A number of algorithms together with implementations have been proposed in the literature (Lehmann and Hitzler, 2010; Fanizzi et al., 2008; Lehmann, 2009; Iannone et al., 2007a). Concept Learning and ILP have a similar relationship as DL and Logical Programming, respectively, as such Concept Learning research has been heavily influenced by the developments within ILP.

When learning a concept, one has as goal of finding a generalized and correct definition of such a concept from a set of examples, as defined below:

Definition 2 (Concept Learning).

$SCL(C_T, \mathcal{E}, \mathcal{KB})$

Given:

- a knowledge base \mathcal{KB} ;
- a target concept C_T such as $C_T \notin \mathcal{KB}$;
- a set of target examples \mathcal{E} , typically divided into positive (\mathcal{E}_p) and negative (\mathcal{E}_n) examples, such that $\mathcal{E} = \mathcal{E}_p \cup \mathcal{E}_n$

Find a definition C for the concept C_T such that:

- $\mathcal{KB} \cup C \models \mathcal{E}_p(\text{complete})$;
- $\mathcal{KB} \cup C \not\models \mathcal{E}_n(\text{consistent})$;
- $C_T \in C_{T_S}$ ⁴, $C_{T_S} = \{C_T, C_{T_1}, \dots, C_{T_n}\}$, $\text{length}(C_T) \leq \text{length}(C_{T_i})$ for $i = 1, \dots, n$ (*parsimony*) ⁵

⁴The set of possible solutions.

⁵Within Concept Learning this is viewed as a general goal more than as a requirement.

Definition 3 (Length). *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.*

Definition 4 follows if a concept definition found is both *complete* and *consistent*.

Definition 4 (Consistency between adding examples and concept definition).

If definition C for the target concept C_T , with the set of examples $\mathcal{E}_T = \mathcal{E}_{T_p} \cup \mathcal{E}_{T_n}$, is found, then $\mathcal{KB} \cup C \cup \mathcal{E}_{T_p} \cup \mathcal{E}_{T_n}$ is consistent and semantically equal to $\mathcal{KB} \cup C$

An algorithm for concept learning is usually composed of the following elements (Lehmann and Hitzler, 2010)

- **a refinement operator** that builds the search tree of concepts;
- **a search algorithm** that controls how this search tree is traversed;
- **a scoring function** that evaluates the nodes of the tree and to point out the best current concept candidate.

The refinement operator allows one to find candidate concept definitions through two basic tasks: generalization and specialization (Lehmann and Hitzler, 2008a). Such operators in both ILP and description logic learning rely on $\hat{\sqsubseteq}$ -subsumption to establish an ordering so as to traverse the search space. If a concept C subsumes a concept D ($D \sqsubseteq C$), then C covers all examples which are covered by D , which makes subsumption a suitable order. Arguably the best refinement operator for description logic learning is the one available in the DL-Learner system-(Lehmann and Hitzler, 2008a, 2010).

In a deterministic setting a cover relationship simply tests whether, for given candidate concept definition (C), a given example e holds; that is, $\mathcal{KB} \cup C \models e$ where $e \in \mathcal{E}_p$ or $e \in \mathcal{E}_n$. In this sense, a cover relationship $\text{cover}(e, \mathcal{KB}, C)$ indicates whether a candidate concept covers a given example (Fanizzi et al., 2008). In DL learning, one often compares candidates through score functions based on the number of positive/negative examples covered. In DL-Learner a fitness relationship considers

2.4 MODEL'S SIMPLICITY AND HOW IT AFFECTS ITS UNDERSTANDABILITY

Historically simplicity has been a hallmark of philosophical and later scientific descriptions. Occam's razor ⁶, or *Lex Parsimoniae* is a logical principle taken as a tenet of the modern philosophy of science. Aristotle's *Posterior Analytics* summarizes the parsimony principle:

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

The parsimony principle is a way to choose between competing hypotheses. In principle the parsimony's primacy cannot be empirically proved, however, its build upon an intuition on the nature of reality.

The modern philosophy of science heirs from the tradition of philosophers such as Plato, Aristotle, and later, Bacon, Descartes, Hume, ... and also holds Occam's razor as an heuristic for discerning competing hypotheses. Therefore, it is a fact, at the very least historical, that simpler hypotheses are preferable to equivalent more complex ones.

The ideas proposed in the philosophy of science are used to create machine learning methods for inductive reasoning (Muggleton and De Raedt, 1994), such as: *Inductive Logic Programming* and *Concept Learning* ⁷. Thus, one can argue that the preference for simpler theories or terminologies should be maintained.

In (Sommer, 1995a) the author present some intuitions on how simplifying a theory can cause it to be more understandable. It does so by appealing to an analogy between logic theories and free text. Textual sentences can be seen as a metaphor for logical predicates. Assuming this relationship, the paper continues to show that within linguistics and psychology (psycholinguistics) there is an inverse relationship between the complexity of the text and its understandability. The paper

⁶Johannes Poncius is the 17th scholar responsible for the classical formulation

⁷Both particularly relevant to this work.

presents a set of intuitions to define how understandable a theory is, this set can be summarized in:

1. A deeper theory is preferable to a shallow one.
 - The depth of a theory is the maximum number of inferences required to answer a goal query.
2. The longer the concept, the harder it is to understand.
3. The more redundant a theory is, the less inclined one would be to accept it.

(Sommer, 1995a) presents another interesting argument as to why understandability should be considered in the machine learning setting. *Artificial intelligent* (AI) systems may be divided into *symbolic* (logical programs, ontologies, ...) and *subsymbolic* (neural networks, genetic algorithms, ...) with regards to inspectability (Smolensky, 1987). In symbolical AI a model consists of symbols meant to be interpreted by humans. As a consequence, when a symbolic model is learned it can be inspected, interpret and understood, and even lead to new insights about the domain. However, with subsymbolic AI a good solution is essentially a black box. Hence, when dealing with symbolic models understandability is a desirable trait.

Simplicity onto itself is worthy goal of machine learning, however it should not be argued that choosing a simpler terminology over a more complex yield a better accuracy (Domingos, 1998). In this work, we have the main goal of learning terminologies that are more understandable (shorter) while still maintaining a comparable accuracy.

3 MULTIPLE CONCEPT LEARNING

In section 2.3 we presented the learning process for a single concept in Definition 2. Given that a concept learning task is defined as $A_C = \langle \mathcal{KB}, \mathcal{E} \rangle$, the formal definition of the terminology learning task is shown in Definition 5.

Definition 5 (Terminology Learning). *Let:*

- $A_{\mathcal{T}}$ be the terminology learning task, where $A_{\mathcal{T}} = \{A_{C_1}, \dots, A_{C_n}\}$.
- A_{C_i} be the concept learning task, $A_{C_i} = \langle \mathcal{KB}, \mathcal{E}_{C_i} \rangle$.
- \mathcal{KB} be the shared knowledge base for all concept learning tasks in $A_{\mathcal{T}}$.
- \mathcal{T} be the terminology after the terminology learning task is completed.

For each $A_{C_i} \in A_{\mathcal{T}}$, find a concept definition C_i according to definition 2. Then,
$$\mathcal{T} = \mathcal{KB} \cup C_1 \cup \dots \cup C_n.$$

When approaching the task of learning a terminology one could use two solution types: i) learning the concepts without including dependencies between concepts (independence assumption) or ii) learn the concepts regarding dependencies, henceforth called *Multiple Concept Learning* (MCL).

Suppose the following problem: we want to learn new concepts descriptions related to an existing terminology of the kinship domain (Fig. 2.2). The set of concepts we wish to learn is $A_{\mathcal{T}} = \{\text{GRANDPARENT}(\text{GP}), \text{GRANDMOTHER}(\text{GM}), \text{GRANDFATHER}(\text{GF})\}$. In order to learn each concept's description we have to define appropriate concept learning tasks for each concept in $A_{\mathcal{T}}$. As previously defined, a concept learning task is composed by a set of examples $\mathcal{E} = \langle \mathcal{E}_p, \mathcal{E}_n \rangle$ and the background knowledge \mathcal{KB} . Because all concepts in $A_{\mathcal{T}}$ belong to the same domain, the \mathcal{KB} can be shared. Suitable sets of examples need to be created using the individuals described in the \mathcal{KB} . After all this, we have defined a terminology learning task $A_{\mathcal{T}}$.

Now we can learn definitions for each concept in $A_{\mathcal{T}}$ independently. However, because of our familiarity with the knowledge domain we know that dependencies between the concepts in $A_{\mathcal{T}}$ exist, for example, a proper definition for GRANDMOTHER can utilize GRANDPARENT.

Table 3.1: Example of the output of a terminology learning task with and without the independence assumption.

Concept	With Independence Assumption	Without
GP	$\exists \text{PARENT}.\exists \text{PARENT}.\top$	$\exists \text{PARENT}.\exists \text{PARENT}.\top$
GM	$\text{FEMALE} \sqcap \exists \text{PARENT}.\exists \text{PARENT}.\top$	$\text{FEMALE} \sqcap \text{GP}$
GF	$\text{MALE} \sqcap \exists \text{PARENT}.\exists \text{PARENT}.\top$	$\text{MALE} \sqcap \text{GP}$

Table 3.1 displays the results for the terminology learning task while upholding the independence assumption, and while regarding the dependencies among the concepts being learned. From this example we can see a suggestion that when the independence assumption is removed, the induced terminology is more compact and readable. This example inspired our main hypothesis: "If dependencies among the concepts of the domain that need to be learned are found then clearer and compact terminologies are yielded while their quality is maintained."

From this main hypothesis we derived tree auxiliary hypotheses regarding the search for dependencies among the concept in a terminology learning task:

1. “If dependencies among the concepts that will be learned are found *before* the learning process then clearer and compact terminologies are yield while maintaining their quality.”
2. “if dependencies among the concepts that will be learned are found *during* the learning process then clearer and compact terminologies are yield while maintaining their quality.”
3. “if dependencies among the concepts are found within a terminology then a clearer and compact version of the terminology can be found while maintaining their quality. ”

In the rest of this chapter we will present a solution for each of the three auxiliary hypotheses. Next a brief summary of the solutions:

- Before the learning phase
 - *Before Terminology Learning*(BTL): Pre processing the the concept learning tasks of a terminology learning task to devise a learning order for the concepts. The concept learned at iteration i can be included in the definition of the concept learned at iteration $i + 1$.
- During the learning phase
 - *During Terminology Learning*(DTL): Changing the learning algorithm or the way it interprets the sets of examples so that a concept’s definition can take into account other concepts.
- After the learning phase
 - *After Terminology Learning*(ATL): Processing a terminology to remove redundancies. This terminology may already exist or be the output of a terminology learning task.

The methods BTL and DTL have a greater relationship to the concept learning process behaviour than ATL, specifically with its refinement operator. Candidate

definitions are generated by the refinement operator using the DLs language constructors and \mathcal{KB} 's content. If, within a terminology learning task, a concept definition that was learned in a previous step is added to \mathcal{KB} it can be used while learning the next concepts. Next we have the definition for the ensuring the shortest concept description within the MCL setting:

Definition 6 (Ensured Shortest Concept Description within MCL). *The shortest concept description for $A_{C_i} \in A_{\mathcal{T}}$, $A_{\mathcal{T}} = \{A_{C_1}, \dots, A_{C_n}\}$, $i = 1, \dots, n$ is:*

$$SCL(C_i, \mathcal{E}_i, \mathcal{KB} \cup_{i=1}^n S_{\mathcal{T}}/A_{C_i}),$$

where:

- $SCL(Target, \mathcal{E}, \mathcal{KB})$ is the function for concept learning, as defined in definition 2
- $S_{\mathcal{T}}$ is the set containing the solution for each $A_{C_i} \in A_{\mathcal{T}}$.
- $S_{\mathcal{T}}/A_{C_i}$ is $S_{\mathcal{T}}$ minus A_{C_i}

When a concept is learned after all the other concepts in the terminology learning task, its learning process will have the largest amount of information compared to any of the other concepts in the terminology learning task, because it will have available the background knowledge and descriptions of all concepts learned in previous iterations. This ensures that any last concept will have the shortest description possible within its MCL setting, because if the best description for a concept contains a group of axioms that are equivalent to any other concept in the terminology learning task, the learning process can replace the group by the concept name, thus yielding a shorter description. However, the best definition for a concept might not depend of all the other concepts in the terminology learning task. Thus, we can define the shortest concept description as:

Definition 7 (Shortest Concept Description within MCL).

The best(A_{C_i}) for $A_{C_i} \in A_{\mathcal{T}}$, $A_{\mathcal{T}} = \{A_{C_1}, \dots, A_{C_n}\}$, $i = 1, \dots, n$ is:

$$SLC(A_{C_i}, \mathcal{E}_i, \mathcal{KB} \cup_{i=1}^n \{S_{\mathcal{T}} \cap best(A_{C_i}).contains\}),$$

where:

- $SCL(Target, \mathcal{E}, \mathcal{KB})$ is the function for concept learning, as defined in definition 2
- $S_{\mathcal{T}}$ is the set of solutions for $A_{\mathcal{T}}$.
- $best(X)$ is the shortest description for concept learning task X .
- $X.contains$ is the set of concepts used in description X .

In the following sections we will present the methods for Multiple Concept Learning without the independence assumption in DL.

3.1 BEFORE TERMINOLOGY LEARNING (BTL)

As seen in definition 7 the shortest solution for a concept learning task is yield when the it is learned after all the concepts it depends on. In this section, we argue that finding out the subsumption relationship between concepts makes it is possible to yield an ordering of the concepts, which in return will support the learning of a rich – although simple and easily understandable – terminology. The question that arises is how to obtain a subsumption order from concepts that do not have definitions yet, i.e., before their definitions are learned. We tackle this problem here by developing a procedure capable of finding out the relations among concepts *before* learning them, by taking advantage of their set of examples.

3.1.1 CONCEPTS DEPENDENCY

When using machine learning techniques to learn concepts, the examples are a mandatory component of the process, they mold the learned concept definition, distinguishing one concept from another. Because of the completeness and consistency properties of the concept learning, the set of positive and negative examples provide strong evidence for the existence of dependencies and relationships among

concepts. Thus, in this section we present a pre-learning phase MCL method based upon the set of examples for discovering concept dependency.

Let $A_{C_i} = \langle \mathcal{KB}, \{\mathcal{E}_{p_i}, \mathcal{E}_{n_i}\} \rangle$ and $A_{C_j} = \langle \mathcal{KB}, \{\mathcal{E}_{p_j}, \mathcal{E}_{n_j}\} \rangle$ be the concept learning tasks for concepts C_i and C_j respectively. After learning these concepts, their definitions will be part of the \mathcal{KB} 's *TBox*. Therefore, a comparative analysis of the individuals belonging to \mathcal{E}_{p_i} and \mathcal{E}_{p_j} points out how strongly related the corresponding concepts are and what might be the best order (\preceq) for learning them. Briefly, we argue that high similarity on the positive examples sets indicates high similarity between concept definitions. Example 3 illustrate this point.

Example 3. Let $A_{C_1} = \langle \mathcal{KB}, \mathcal{E}_1 \rangle$, $\mathcal{E}_1 = \langle \mathcal{E}_{p_1}, \mathcal{E}_{n_1} \rangle$ and $A_{C_2} = \langle \mathcal{KB}, \mathcal{E}_2 \rangle$, $\mathcal{E}_2 = \langle \mathcal{E}_{p_2}, \mathcal{E}_{n_2} \rangle$. Also, $\mathcal{E}_{p_1} = \{john, bob, edd\}$ and $\mathcal{E}_{p_2} = \{bob\}$ be the set of positive examples for concepts FATHER and GRANDFATHER, respectively. $\mathcal{E}_{p_2} \subseteq \mathcal{E}_{p_1}$, then the definition of concept FATHER will include more individuals than the definition of concept GRANDFATHER, thus being more general. Therefore, we argue that in the GRANDFATHER's definition the concept FATHER can be used, but not the other way around. Thus, if FATHER is learned and added to \mathcal{KB} before learning GRANDFATHER, then FATHER can be used to compose the definition of GRANDFATHER. A learning order is then established between the two concepts ($FATHER \preceq GRANDFATHER$).

The comparative analysis between two concepts (C_i and C_j) through their sets of positive examples comprises the following cases (Fig. 3.1 illustrates these cases):

1. Identical sets ($C_i \equiv C_j$): all elements belonging to one set also belong to the other set;
2. Disjoint sets ($C_i \cap C_j = \emptyset$): both sets have no elements in common;
3. Inner set ($C_i \subseteq C_j$): one set is a subset of the other;
4. High intersection: the proportion of shared elements overcomes a threshold;

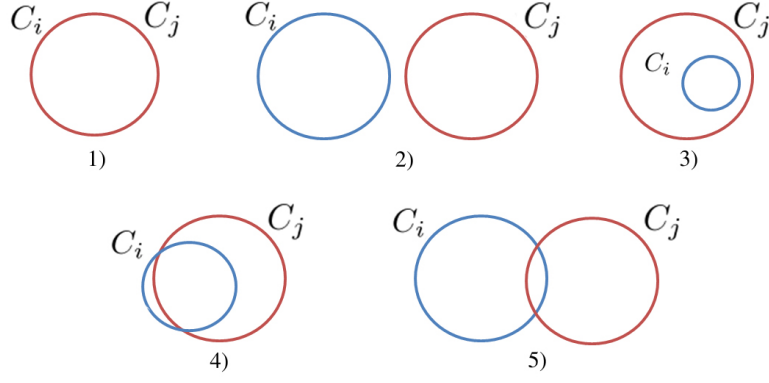


Figure 3.1: The relationship among concepts C_i and C_j through their positive examples sets, represented as Venn diagrams.

5. 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 1 describes a upper limit, where the concepts are totally related, while case 2 indicates a lower limit, where the two concepts have no relationship. Furthermore, cases 1, 2 and 3, are the ideal cases because the underlying relationships between concepts are clear; on the other hand, the cases 4 and 5, 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 3 is a special case of 4, when the threshold is equal to 1; the case 2 is a special case of 5, 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 A_T . The resulting explicit relations among the concepts are translated to a subsumption tree (*a taxonomy*). For instance, consider the taxonomy depicted in Figure 3.2 for domain \mathcal{D} and set of concepts $N_C = \{C_1, C_2, C_3, C_4, C_5\}$.

The root node indicates the set of all individuals of the domain \mathcal{D} . A concept in a branch on level i is more general than a concept on the same branch in level j , thus concept C_1 is more general than concept C_3 . Assuming that a more general

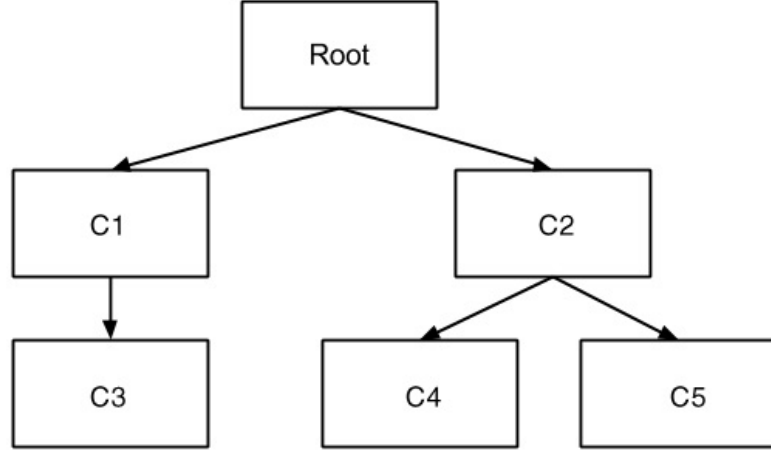


Figure 3.2: An example of a taxonomy depicting the relation among concepts C_1, C_2, C_3, C_4 and C_5 .

concept can be used in the definition of a more specialized one, the latter should be learned first. Thus, the resulting taxonomy of the concepts can be used to determine the best ordering to learn the concept definitions.

3.1.2 LEARNING CONCEPT ORDERING

In Section 3.1.1, five possible results were detected after analyzing the relations between two concepts (C_i and C_j) through their sets of positive examples. From the first three cases, it is possible to directly acquire the relationship between the concepts. The two remaining cases requires a further analysis, aiming at deciding whether a subsumption relation is assumed or a total independence between C_i and C_j is a better choice. In this section, a procedure concerning the five possibilities is devised, which will lead to a taxonomy of concepts.

Algorithm 1 presents the top-level procedure for building a taxonomy of concepts that will be used to define an order for learning the concepts of a terminology.

The algorithm receives as input a terminology learning task, as outlined in Definition 5, 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 father of all the concepts in $A_{\mathcal{T}}$.

Algorithm 1 Algorithm for learning a taxonomy of concepts

Require: a terminology learning task ($A_{\mathcal{T}} = \langle A_{C_1}, A_{C_2}, \dots, A_{C_n} \rangle$, where all $A_{C_i} = \langle \mathcal{KB}, \mathcal{E}_i \rangle$ is a concept learning task for concept C_i)

Ensure: a taxonomy of concepts \mathcal{T}

- 1: Starts \mathcal{T} with a default concept in the root;
 - 2: Add each concept C_i considered in $A_{\mathcal{T}}$ as a child of the root;
 - 3: **for** each $A_{C_i}, A_{C_j} \in A_{\mathcal{T}}$ **do**
 - 4: $\mathcal{T} \leftarrow$ call Algorithm 2 with $A_{C_i}, A_{C_j}, \mathcal{T}$
 - 5: **end for**
 - 6: **return** \mathcal{T}
-

This algorithm starts by creating a tree with only the root. Next, each concept (C_i) corresponding learning task (A_{C_i}), where $A_{C_i} \in A_{\mathcal{T}}$, is included in the tree as child of the root. Then, Algorithm 2 is called for each pair of concepts C_i and C_j in order to find their relationship. After checking each possible dependency, the final taxonomy of concepts is returned.

In Algorithm 2 the relationship between the positive examples (\mathcal{E}_p) sets of the two concepts are analyzed in order to resolve in which of the five cases discussed in Section 3.1.1 it belongs. The algorithm begins by creating a variable that holds the amount of shared individuals between the example sets (line 1). The first case checked (line 2) is the one with two equivalent concepts, if that is the case the taxonomy is not modified, remaining the same as the previous iteration. In this way, both concepts are maintained unchanged in the resulting taxonomy, which makes possible to learn a proper definition for each one of them, even though they share the same set of positive examples. An alternative choice would be to keep only one of them in the taxonomy and later indicate in the terminology their equivalence. We chose the former because it is generally the case that the negative examples could be responsible for pointing out a different definition for each one of them.

Next, the algorithm verifies whether the concepts are disjoint (line 6). If this is the case, then there is no relationship between the concepts and the taxonomy is not changed. In case the set of positive examples associated to one concept is a subset of the other (line 9, 13), a relationship between them is included in the taxonomy.

The *subsumee* concept is put as a child of the *subsumer* in the tree. Moreover, the edge connecting the *subsumee* concept and the root is removed, as it is no longer necessary.

The two remaining decision are made according to the size of the intersection set between the two concepts. To decide whether the intersection is high or low, we employ a threshold parameter, that needs to be set before starting the process. The amount of intersection is computed by selecting the concept with less examples and dividing the number of shared individuals by its examples set size (line 24). In case this value is higher than the threshold, the procedure assumes that a relationship exists and the concept with more examples is included in the taxonomy as the *subsumer*, while the concept with less examples is included as its *subsumee* (line 25); again the relationship between root and the *subsumee* is removed. An important thing to notice is that only the root is removed as a parent of the *subsumee* concept; all the other parents it may have are maintained, such that the right order can be obtained.

3.1.3 LEARNING TERMINOLOGY THROUGH CONCEPT ORDERING

Algorithm 3 presents the overall procedure for learning a terminology. It requires as input the terminology learning task $A_{\mathcal{T}}$. As in our method it is necessary to find a taxonomy tree, so that a concept ordering is induced from such a tree, the first step of the algorithm is to call the procedure devised as Algorithm 1, which is going to return the tree comprising each concept addressed by the terminology problem.

Next, it is necessary to establish the order that the algorithm is going to follow to learn each concept description. This is done by traversing the tree in a breadth-first manner, and collecting all nodes at a level in order. For instance, considering the taxonomy in Figure 3.2, the concepts would be learned in the following order $< C_1, C_2, C_3, C_4, C_5 >$. In this way, it is possible to guarantee that a concept is only handled after the descriptions for all its parents have been learned. This method

Algorithm 2 Algorithm for Ascertaining Dependencies among Concepts in a Terminology

Require: Concept learning tasks $A_{C_i} = \langle \mathcal{KB}, \{\mathcal{E}_{p_i}, \mathcal{E}_{n_i}\} \rangle$ and $A_{C_j} = \langle \mathcal{KB}, \{\mathcal{E}_{p_j}, \mathcal{E}_{n_j}\} \rangle$; a ordering tree \mathcal{T} .

Ensure: An updated ordering tree \mathcal{T}

```

1: sharedIndividuals  $\leftarrow 0$ 
2: if  $A_{C_i} \equiv A_{C_j}$  then
3:   return  $\mathcal{T}$ 
4: end if
5: sharedIndividuals  $\leftarrow |\mathcal{E}_{p_i} \cap \mathcal{E}_{p_j}|$ 
6: if sharedIndividuals = 0 then
7:   return  $\mathcal{T}$ 
8: end if
9: if  $\mathcal{E}_{p_i} \subseteq \mathcal{E}_{p_j}$  then
10:   include  $A_{C_j}$  as father of  $A_{C_i}$  in  $\mathcal{T}$ 
11:   remove ROOT from  $A_{C_i}$  parents set
12: end if
13: if  $\mathcal{E}_{p_j} \subseteq \mathcal{E}_{p_i}$  then
14:   include  $A_{C_i}$  as father of  $A_{C_j}$  in  $\mathcal{T}$ 
15:   remove ROOT from  $A_{C_j}$  parents set
16: end if
17: if  $|\mathcal{E}_{p_i}| \leq |\mathcal{E}_{p_j}|$  then
18:   smaller_concept  $\leftarrow A_{C_i}$ 
19:   larger_concept  $\leftarrow A_{C_j}$ 
20: else
21:   smaller_concept  $\leftarrow A_{C_j}$ 
22:   larger_concept  $\leftarrow A_{C_i}$ 
23: end if
24: if (sharedIndividuals / |smaller_concept|  $\geq$  threshold) then
25:   include larger_concept as father of smaller_concept in  $\mathcal{T}$ 
26:   remove ROOT from smaller_concept parents set
27: end if
28: return  $\mathcal{T}$ 

```

Algorithm 3 Top-Level Algorithm for Learning Terminologies

Require: A terminology learning task $A_{\mathcal{T}}$.

Ensure: A Terminology \mathcal{TE}

- 1: find a Taxonomic Tree \mathcal{T} through Algorithm 1
 - 2: find an Ordered Sequence $A_{\mathcal{S}}$ of concept learning tasks $A_{\mathcal{T}}$ using \mathcal{T} , where concepts only come after their parents
 - 3: **for** each $A_{C_k} \in A_{\mathcal{S}}$, in the established order **do**
 - 4: learn a description C_k from A_{C_k} , using any algorithm that learns concepts in Description Logics
 - 5: include C_i in \mathcal{KB} for all $A_{C_i} \in A_{\mathcal{S}}$ where $i > k$
 - 6: **end for**
 - 7: **return** \mathcal{TE}
-

allows for the discovery of unpredicted relationships because as ordering is devised between concepts in the same taxonomic level. Different treatment of same level concepts could be devised, however, this is not the main goal of the taxonomy.

Finally, each concept is learned gradually in order, through a loop that visits the concept learning task in the order defined in the second main step of the algorithm. To learn a description, the concept itself, its positive and negative examples and the current background knowledge are submitted to a description logic learning algorithm, such as DL-Learner (Lehmann and Hitzler, 2010), DL-FOIL (Fanizzi et al., 2008) or Yin-Yang (Iannone et al., 2007a). Then, the description found is included in the TBox of each \mathcal{KB} of the following A_{C_i} , so that it can be used in next iterations of the loop, as part of other descriptions. In this way, the learning algorithm can learn concept's definitions according to the dependencies found in in Algorithm 2, in a rather ideal order discovered from the tree learned by the Algorithm 1. The algorithm ends by returning the terminologies learned at each step of the loop.

3.2 DURING TERMINOLOGY LEARNING (DTL)

As shown in definition 6 the shortest description for a single concept learning task within a terminology learning task is obtained when it is learned after all the other

concepts, or when it is learned after all concepts it might depend on (definition 7). In summary it must find an answer for:

“What is the learning order that yields the shortest terminology?”

From definition 6, if it were possible to guarantee $S_{\mathcal{T}}/A_{C_i}$ for any i we could find the best possible solution for the terminology \mathcal{T} ¹. On this section we provide a different view on the terminology learning problem, arguments and scenarios where it would be possible to obtain $S_{\mathcal{T}}/A_{C_i}$ for any i . Furthermore, if that is possible, then the concept learning mechanism will be the sole responsible for the dependencies.

The question that arises is: “how can the definition for C_j be found before being learned?” In the remainder of this section we argue that under some conditions is possible to obtain a “definition” for C_j previous to its learning. Definition 8 underline the set of conditions under which this is possible.

Definition 8 (DTL’s Restrictions). *The following are restrictions for the proper work of the method DTL:*

1. *The background knowledge is a finite model.*
2. *For every concept C in terminology learning task \mathcal{T} and every individual I relevant to the concept C , $I \in \mathcal{E}_{P_C}$ or $I \in \mathcal{E}_{N_C}$.*

The domain owner has to investigate if it is possible or desirable to uphold restriction 2, naturally upholding this restriction may be reasonable because the amount of information on \mathcal{E} tends to yield better results.²

When DTL’s restrictions are uphold, because of the idea presented in definition 4, its possible to use the sets of examples \mathcal{E}_i of each concept learning task within a terminology learning task $A_{\mathcal{T}} = A_{C_1}, \dots, A_{C_n}$, as a surrogate for the definition that will be found later with no semantic lost. There is no difference with regards to the result obtained from the reasoning services for a query $C(a)$, if it is an assertion (ABox) or this follows from the terminology (TBox.)

¹Contingent upon SCL returning the best solution.

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

This has an added benefit that each concept learning task can be executed independently, allowing for parallel learning of each concept.

The example 4 shows the output of the terminology learning task for the set of concepts $\{\text{GRANDPARENT}(GP), \text{GRANDMOTHER}(GM), \text{GRANDFATHER}(GF)\}$. With DTL's local goal is possible to find the shortest description for each concept (notice GP). This spurs the rise of a problem as depending on the set of concepts, cycles may be formed ³. This goes against one of the general assumption we need to uphold - concept definitions are acyclic (section 2.1). We propose two approaches to deal with cycles: (i) restrict the set of examples added to \mathcal{KB} ; (ii) remove the cycles on a post procedure.

Example 4. *Let \mathcal{KB} be as presented in Figure 2.2, $A_{\mathcal{T}} = \{GP, GM, GF\}$. Executing DTL we have the resulting terminology \mathcal{T} :*

- $GP \equiv GM \sqcup GF$
- $GF \equiv GP \sqcap male$
- $GM \equiv GP \sqcap female$

(i) If a concept C_j of the concepts in $A_{\mathcal{T}}/C_i$, $i \neq j$, already has a definition and it uses C_i do not include C_i or C_j in \mathcal{KB} . This prevents that any new definition creates a cycle. As a consequence, and 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, identify the concepts that incurred in cycles and relearn them, avoiding the current or a recurring cycle. Chose the definitions that yield smaller definitions.

The second method have a better chance to return the optimal acyclic solution, but may relearn several concepts, thus the first method is likely to be more efficient. In both methods the best acyclic solutions is not guaranteed.

³The definition for GP makes a cycle.

3.2.1 DTL WITH CYCLE REMOVAL

As stressed earlier on this chapter the normal DTL method has a great probability of inducing terminologies containing cycles. As one of the possible approaches to deal with cycles is to remove them after the terminology is obtained, the rest of this section will present such method.

The proposed method will only deal with cycles with depth⁴ of one, Example 5 displays a terminology with a cycle of depth one. However, the following algorithms can be modified to work with deeper cycles if a cycle checking is defined and alternative definitions⁵ for all concepts in the cycle are found. We left the implementation of deeper cycle removal for a future work.

Example 5. *The terminology containing the concepts: $A \equiv B \sqcap C \sqcap D$ and $B \equiv F \sqcap D \sqcup A$, has a cycle with depth of one, because the concept A appears in the definition of B while it uses B within its definition, needing only one inferential step before incurring in a cycle.*

The Algorithm 4 is the entry point for the learning process. It runs the DTL for the terminology learning task and call the Algorithm 5 to remove the cycles of the learned terminology.

Algorithm 4 DTL with Cycle Removal Main algorithm

Require: A terminology learning task $A_{\mathcal{T}}$.

Ensure: A Terminology \mathcal{TE}

- 1: **for** each $A_{C_i} \in A_{\mathcal{T}}$ **do**
 - 2: Add $A_{\mathcal{T}}/A_{C_i}$ example sets into \mathcal{KB}'
 - 3: Change A_{C_i} 's \mathcal{KB} to \mathcal{KB}'
 - 4: Learn a description C_i for A_{C_i}
 - 5: **end for**
 - 6: $\mathcal{TE} \leftarrow$ call Algorithm 5($A_{\mathcal{T}}$)
 - 7: **return** \mathcal{TE}
-

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

⁵The alternative definitions should be constructed in a way to prevent the same recurring cycle.

The Algorithm 5 is used to remove the cycles of a set of concept learning task. It starts by defining an empty map of blocked concepts. A concept C_j is blocked concept for the learning of a concept C_i , if in a previous iteration there was a cycle between C_i and C_j and C_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. In this work, we will only check for cycles with the depth of one⁶. 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 13, 14). 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 17). The choice is made in favor of the alternative definition with the smallest impact on the terminology, i.e., the one with least difference in length with regards 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 AFTER TERMINOLOGY LEARNING (ATL)

The technique to compact existing definitions proposed in this section 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*.

In (Melo et al., 2014) this approach received the name *R2D2* for “Redundancy

⁶See example 5

Algorithm 5 Remove cycles from learning tasks algorithm

Require: A set of learning task $A_{\mathcal{T}}$ with definitions.

Ensure: A Terminology \mathcal{TE}

```
1: mapBlockedConcepts  $\leftarrow \emptyset$  {Contains the set of concepts blocked from the learn-
   ing of each concept}
2: conceptsToVerify  $\leftarrow A_{\mathcal{T}}$ 
3: while conceptsToVerify  $\neq \emptyset$  do
4:    $A_i \leftarrow \text{conceptsToVerify.pop}$ 
5:    $A_j \leftarrow \text{null}$ ;
6:   for each  $A_p \in \text{conceptsToVerify}$  do
7:     if There is a cycle between  $A_i$  and  $A_p$  then
8:        $A_j \leftarrow A_p$ 
9:       break from the for loop
10:    end if
11:  end for
12:  if  $A_j \neq \text{null}$  then
13:     $\mathcal{KB}_{A_i} \leftarrow A_{\mathcal{T}} / \{A_j \cup \text{Dependents of } A_j \cup \text{mapBlockedConcepts of } A_i\}$ 
14:     $\mathcal{KB}_{A_j} \leftarrow A_{\mathcal{T}} / \{A_i \cup \text{Dependents of } A_i \cup \text{mapBlockedConcepts of } A_j\}$ 
15:    Learn a description  $D_i$  for  $A_i$ , using any algorithm that learns concepts
    in DL with  $\mathcal{KB}_{A_i}$ 
16:    Learn a description  $D_j$  for  $A_j$ , using any algorithm that learns concepts
    in DL with  $\mathcal{KB}_{A_j}$ 
17:     $A_b \leftarrow A_i$  or  $A_j$  by the best description between  $D_i$  and  $D_j$ 
18:    add  $A_b$  to conceptsToVerify
19:    add the other concept to the blocked concepts of  $A_b$ 
20:  end if
21: end while
22: return  $\mathcal{T}$ 
```

Removal on DLs Descriptions”. In the remainder of this work, we will interchangeably refer to this method as R2D2 or ATL.

3.3.1 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 6 illustrate a possible syntactic compression.

Example 6. 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 .

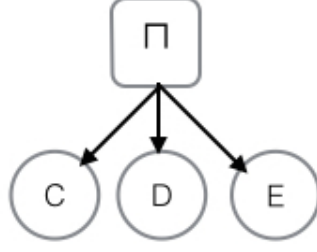


Figure 3.3: Example 8's syntactic tree representation.

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's constructors are commutable, as shown in Example 7.

Example 7. *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 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 8.

Example 8. *Let A, B, C, D, E concepts in a terminology, such that concept $A \equiv C \sqcap D \sqcap E$ and $B \equiv E \sqcap C \sqcap D$. Fig. 3.3 shows the representation for both concepts.:*

Fig. 3.4 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.

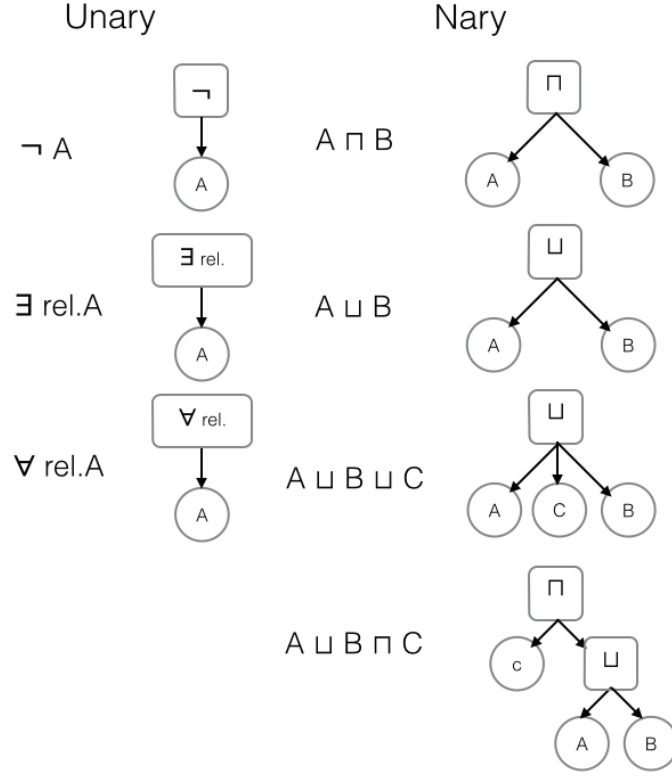


Figure 3.4: The syntactical tree representation of \mathcal{ALC} 's constructors.

3.3.2 REDUNDANCY REMOVAL ON DLS DESCRIPTIONS (R2D2)

Algorithm 6 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.

It starts by sorting the set of concepts in ascending order of length (Definition 9). This is an important step, because a concept can only have a substitution involving a smaller concept.

Definition 9. *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 7 and 8, that we discuss later. Once it has the tree representation of all concepts, it can proceed to find

Algorithm 6 Syntactical Compression Main Algorithm

Require: a set of concept definitions $C_d = \{C_1, \dots, C_n\}$ and the set of valid constructors in the DL

Ensure: a set of compressed concepts definitions $C'_d = \{C'_1, \dots, C'_n\}$

```
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 7( $C_i$ , 0,  $C_i$  length, valid unary constructors, valid nary constructors);
5: end for
6: for each pair  $\mathcal{T}_{ci}$  and  $\mathcal{T}_{cj}$  in  $\mathcal{T}_c$ , where  $i < j$  do
7:   Algorithm 9( $\mathcal{T}_{ci}$ ,  $\mathcal{T}_{cj}$ );
8: end for
9: for each  $\mathcal{T}_{ci} \in \mathcal{T}_c$  do
10:   $C_i \leftarrow \text{TreeToDefinition}(\mathcal{T}_{ci})$ ;
11: end for
12: return  $C_d$ 
```

suitable substitutions. It does so by exhausting all the pairs of concepts with a smaller concept and a larger one (line 6), and calling Algorithm 9 (line 7) 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 10). The translation method is done via a pre-order traversal in the tree.

SYNTACTIC TREE CONSTRUCTION

Algorithms 7 and 8 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 22 of algorithm 7 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 7 Logical Description to Syntactical Tree

Require: a concept description C ; a lower bound; an upper bound; a set of unary constructors $S_u = U_C1, \dots, U_Cn$ and a set of nary constructors $S_n = N_C1, \dots, N_Cn$

Ensure: A syntactical Tree *root*

```
1: create an empty root;
2: create the auxiliary 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  $\leftarrow$  Algorithm 7( $C$ , 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:    end if
13:    increment index;
14:    continue to next loop iteration;
15:  end if
16:  if currentConstruct  $\in S_n$  then
17:    if root is empty then
18:      root  $\leftarrow$  currentConstruct;
19:      if firstConcept is not empty then
20:        add firstConcept as a child of currentConstruct;
21:      end if
22:      if root  $\neq$  currentConstruct then
23:        add root as a child of currentConstruct;
24:        root  $\leftarrow$  currentConstruct;
25:      end if
26:    end if
27:    increment index;
28:    continue to next loop iteration;
29:  end if
30:  if currentConstruct  $\in S_u$  then
31:    currentConstruct  $\leftarrow$  Algorithm 8;
32:    index  $\leftarrow$  end index returned in line 31;
33:  end if
34:  if root is empty then
35:    firstConcept  $\leftarrow$  currentConstruct;
36:  else
37:    add currentConstruct as a child of root;
38:  end if
39: end while
40: if root is empty AND firstConcept is not empty then
41:   root  $\leftarrow$  firstConcept;
42: end if
43: return root
```

Algorithm 8 Recursive Method for Unary Constructors

Require: a concept description C ; a lower bound; a set of unary constructors $S_u = U_C1, \dots, U_Cn$ and a set of nary constructors $S_n = N_C1, \dots, N_Cn$

Ensure: A syntactical Tree *node* and the ending index

```
1:  $index \leftarrow lowerbound$ ;
2:  $currentConstruct \leftarrow C[index]$ ;
3: if  $currentConstruct \in S_u$  then
4:   if  $currentConstruct$  is a unary quantifiers then
5:      $currentConstruct \leftarrow$  all items from index to relation statement;
6:   end if
7:    $currentConstruct.descendent \leftarrow$  Algorithm 8;
8:    $index \leftarrow$  Algorithm 8 index;
9: else
10:  if  $currentConstruct$  is a block then
11:     $currentConstruct \leftarrow$  Algorithm 7( $C$ , index, end block index,  $S_u$ ,  $S_n$ );
12:     $index \leftarrow$  end block's index;
13:  end if
14: end if
15: return  $currentConstruct$  and  $index$ 
```

FINDING PROPER SUBSTITUTIONS

Algorithm 9 is used to perform substitutions between two trees. It receives 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). An equivalent tree is found when: i) all the internal nodes in the smaller tree have an equivalent node in the larger tree subtree (also in the right “position”), and ii) the subtree root of the larger tree contains all the nodes below the smaller tree root. Example 9 shows why the second restriction 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*.

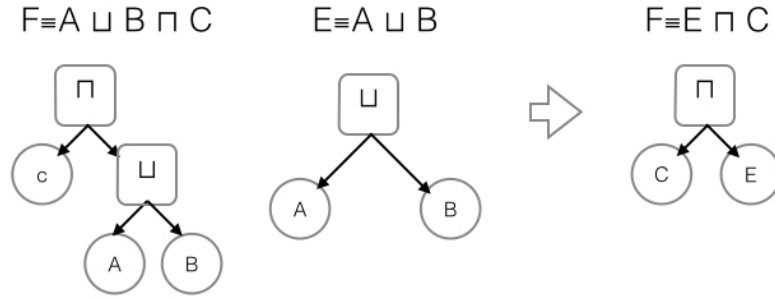
Example 9. Let $F \equiv A \sqcup B \sqcap C$ and $E \equiv A \sqcup B$. Executing the substitution on the tree representation of F and E we have:

Algorithm 9 Syntactical Compression's Substitution

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

```

1:  $nodesToVerify \leftarrow \{\mathcal{T}_l.root\}$ ;
2: while  $nodesToVerify$  is not empty do
3:    $rootLarger \leftarrow nodesToVerify.pop$ ;
4:   if  $rootLarger \equiv \mathcal{T}_s.root$  then
5:      $rootLarger \leftarrow \mathcal{T}_s$  concept;
6:   end if
7:   add  $rootLarger$  in  $nodesToVerify$ ;
8: end while
  
```



3.4 SUMMARY OF METHODS

In this section we will describe the positive and negative aspects of each of the proposed methods for Multiple Concept Learning.

The BTL's method finds a reasonable learning ordering based on the subsumption relationship without incurring in a prohibitive computational cost. The negative side of BTL is its strong bias for dependencies of the subsumer/subsumee type, also, once a relationship is found the order assumes that a concept with a smaller set of examples can be properly defined using the larger in conjunction (\cap) with other description (C).

The DTL's method does not have any bias for a particular type of relationship, because the learning mechanism is the sole responsible for finding and expressing dependencies. Furthermore, the concept learning tasks in a terminology learning task can be executed in parallel. However, the basic DTL method has a serious problem, it does not uphold the DL assumption that states that terminologies should

be acyclic. We presented two solutions for this problem: (i) avoiding the cycles, or (ii) removing the cycles. The second solution, *cycle removal*, have a better chance to return the optimal acyclic solution, but may relearn several concepts, thus the first method, *cycle avoidance*, is likely to be more efficient. In both methods the best acyclic solutions is not guaranteed.

Because they use the sets of examples and the shared individuals within these sets to inform the relationships found, both methods, BTL and DTL, have a strong premise related to the completeness of the examples. Usually, there are no rules related to how one chooses the set of examples for the learning task, it only needs to contain relevant individuals for the concept being learning, there are no indications of how significant this sample of the domain should be.

For BTL and DTL, even when assuming the background knowledge (\mathcal{KB}) is shared by the concept learning tasks in a terminology learning task, there is no guarantee that the same set of individuals will be used as examples for all concept learning tasks. If this is the case, BTL and DTL methods would fail at finding relationships because they use a comparative analysis of the shared individuals to inform the dependencies found.

Therefore, to assure the quality of the dependencies found, we added the following restrictions: i) all concept learning tasks in a terminology learning task must relate to the same knowledge base; ii) all concept learning task examples must relate to the same set of individuals. In this case, all the individuals belonging to a subset of \mathcal{KB} should be classified as either a positive or negative example⁷. This burden may be too big for the domain experts.

The ATL’s solution finds all syntactical redundancies and is independent of the presence of ABoxes, it uses only the terminological descriptions to find the dependencies, because of this, it can be applied to any existing terminology, therefore, also to terminologies induced via a terminology learning task. The method is also DL language independent, as long as it is possible to define if a constructor of the

⁷See Appendix A for a further reading about the relationship of OWA and CWA to inductive and inferential reasoning.

Table 3.2: Matrix of advantages and disadvantage for BTL, DTL and ATL.

	BTL	DTL	ATL
PROS	<ul style="list-style-type: none"> – Reasonable learning order without prohibitive computational cost 	<ul style="list-style-type: none"> – No relationship bias – SCL defines relations – Parallel execution 	<ul style="list-style-type: none"> – Language independent – Finds all syntactical redundancies – Not restricted to terminology learning – ABox independent
CONS	<ul style="list-style-type: none"> – Conjunctive bias – CWA of \mathcal{E} 	<ul style="list-style-type: none"> – CWA of \mathcal{E} – Restriction 2 might be too strong – Must deal with cycles 	<ul style="list-style-type: none"> – Exponential

language is a unary or n-ary constructor. However, ATL's drawback is that the syntactic compression method is exponential.

The table 3.2 summarizes the information presented in this section.

4 EXPERIMENTS

To validate our hypotheses we use the Experimental method. This chapter presents the experimental methodology, it is divided in three sections. The first one describes the set of datasets used throughout the experiments. The second section describes the experiments pertaining each proposed method (BTL, DTL and ATL). The third and last section describes the experiments to compare all the proposed methods and single concept learning.

4.1 DATASETS

The following are the problems used in the experiments.

- **Kinship:** defines family relations through a very simple terminology, containing only two roles (*sibling* and *married*) and three atomic concepts (*gender*, *male* and *female*). The set of concepts used for learning was {*Grand Parent (GP)*, *Grand Father (GF)*, *Grand Mother (GM)*}, obeying their regular semantic. Each concept in this set is interdependent of the others, i.e., one could generate a proper concept definition using a combination of the remaining ones, e.g., $GM \equiv GP \text{ AND } female$, or $GM \equiv GP \text{ AND NOT } GF$.

- **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 large terminology composed of 25 primitive concepts and 20 non-primitive concepts. Additionally, the ABox has assertions for 202 individuals. However, not all the assertions about non-primitive concepts are represented, being necessary inference to achieve them. Since this is a more elaborate dataset than the Kinship, it can lead us to interesting conclusions about the proposed methods. Towards that, two experiments were done with this dataset. In the first one (Exp1), the concepts chosen to be learned were $\{guilty, blameworthy, responsible, notaccident\}$, 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 interdependencies, other set of concepts could have been chosen, those were chosen arbitrarily.
- **OAEI campaign**: ATL's method can deal with terminologies that do not have individuals (ABox) because it works only with concept's descriptions (TBox). Because of this we could devise a different more broader set of problems to validate ATL's approach. We devised a validation experiment using a larger set of problems, containing a number of datasets of OAEI ¹'s 2013 campaign of ontology matching and all datasets packaged as examples on 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.

¹<http://oaei.ontologymatching.org/>

²<http://dl-learner.org/Projects/DLLearner>

4.2 EXPERIMENTS FOR EACH INDIVIDUAL METHOD

4.2.1 BTL: FINDING THE BEST THRESHOLD VALUE

BTL is a method that creates an order over the concepts that a multiple concepts learning algorithm should follow. It uses a threshold parameter that defines a lower limit for the proportion of intersection between two concepts, after which the concepts are considered dependent by the method. In (Melo et al., 2013b) the threshold value was arbitrarily set to 80%. Here, we would like to empirically decide which is the best value for the threshold parameter.

The issue that arises here is what “best” means in this context. The BTL method analyses the concept’s set of positive examples two by two and check whether the proportion of shared examples over the amount of examples within each set is higher than the threshold. This information is then used to build the taxonomy, which in turn is used to define the learning order. Thus, the best threshold value would be the one in which the taxonomy found by BTL is identical to a taxonomy built from the relationships found on the learned terminology, i.e., the relationships predicted by the induced taxonomy should also appear in the learned terminology.

Thus, to analyse the behaviour of BTL approach according to the threshold parameter, we set it to 20%, 40%, 60%, 80% and 90% and computed the matching relationships using the *moral reasoning dataset*. Table 4.1 presents the average predictive results we found, the second column presents the threshold’s value, the values were aggregated if more than one result were the same, where the third column shows the number of predictions returned by the induced taxonomy, the fourth column shows the subset size of relationships that were indeed considered in the learned terminology and the fifth column shows the number of found dependencies that matched those presented in the target description, we called this metric *Hits*. The results indicate that indeed the arbitrarily set threshold is a good value, followed closely by 90%. As it should be expected, requiring only a low intersection

Table 4.1: BTL: Best Threshold Experiment.

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

between two sets of examples yields a high number of predictions that are not actually established during the learning phase. An interesting phenomenon occurred in the problem “exp2” with thresholds of 80% and 90%, in these scenarios relationships different from those of the target concepts were found, this indicates that the terminology learning task chosen has a wide variety of possible relationships with valid descriptions.

4.2.2 DTL

DTL adds the learning examples to the background knowledge in order to induce the shortest solution for a single concept learning task (Definition 7). We devised an experiment using the kinship toy example in order to validate this claim. In this experiment we will not uphold the assumption that the terminology must be acyclic, we did this because we would like to know if DTL’s surrogates are indeed correct and can yield the shortest description for all the concepts in a terminology learning task. Table 4.2 shows the result for this experiment contrasting with the result achieved with the SCL method.

DTL achieves a terminology with the length 9, while SCL found a terminology with the length of 19. Furthermore, DTL found the shortest possible solution for all three concepts in the terminology learning task.

Table 4.2: DTL’s validation experiment results.

Experiment	Concept	Definition	Length
SCL	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
DTL	GP	(grandfather OR grandmother).	3
	GF	(grandparent AND male).	3
	GM	(grandparent AND female).	3

Table 4.3: ATL: Results of the validation experiment.

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 NCL_smallOverlapping_fma	1964	17046	17046	55.59373
OAEI SNOMED_smallOverlapping_fma	2882	35552	35552	86.63409
OAEI NCL_smallOverlapping_snomed	6851	74191	74183	684.26991
OAEI NCL_while_ontology	10280	151316	151308	1997.30955
Rest of datasets (#28)	0	0	0	0

4.2.3 ATL

Different from the BTL and DTL, ATL (R2D2) can be used outside the terminology learning problem, because it is a method for theory restructuring.

In order to evaluate we used the R2D2 validation dataset described earlier. Table 4.3 presents the results obtained from R2D2 for each one of the 46 datasets. Most of the datasets have terminologies with only atomic concepts and roles (ABox), in these cases the method does not propose any replacements on the original terminol-

ogy 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 happened 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) `NCL_small_overlapping_fma.owl`; iii) `NCL_whole_ontology.owl`. Next we analyse each substitution.

conf_Cocus: the original terminology was 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, as an example resulting in: *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.

NCL_small_overlapping_fma.owl and *NCL_whole_ontology.owl*: both terminologies pertain to the same domain of biomedicine, sharing much of the concepts and definitions. Looking closely we can see that the terminology `NCL_small_overlapping_fma.owl` is a subset of `NCL_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 one entertain all possible substitutions. These experiments suggests that the use of R2D2 has the potential to yield terminologies as simple as syntactically possible, provided that the computational cost is not too high.

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 evaluation experiment ten times and calculated the average execution time for each terminology. Fig. 4.1 shows the relationship between the number of concepts/size of terminology and the execution time.

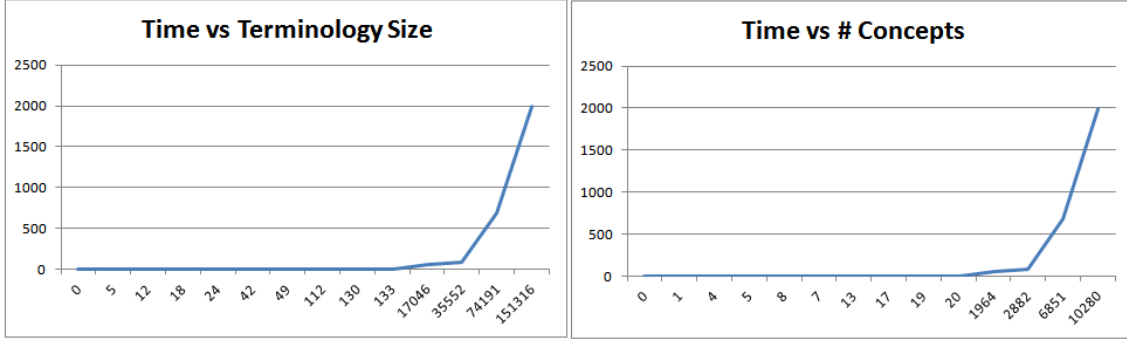


Figure 4.1: R2D2’s Execution time vs Number of concepts and Terminology size.

It is important to stress that, although it seems that there would be an exponential increase in the execution time as terminologies gets larger than the biggest tested, is highly unlikely that terminologies would have much more than 10280 concepts and 151316 in size. Having the 10280/151316 as a soft limit for the complexity, the 20 to 30 minutes of execution time seems reasonable.

4.3 EXPERIMENTS COMPARING METHODS

In this section, we describe experiments conducted towards comparing BTL, DTL and ATL. The experimental design consisted of the following steps:

1. select a problem that has an associated terminology with interdependent concepts;
2. select a set of concepts within such a terminology to compose the concept learning tasks (A_{C_i});

3. define the knowledge base (\mathcal{KB}) and the sets of examples (\mathcal{E}_{p_i} , \mathcal{E}_{n_i}) for all 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 according to the concept;
4. run BTL, DTL, ATL and DL-Learner as a standard single concept learning approach for baseline comparison.
5. analyze the results achieved.

Towards encompassing the first step, we used the datasets *Kinship* and *Moral Reasoner*.

Each dataset was used to evaluate BTL, DTL and ATL from different perspectives. Therefore, a particular experimental methodology was used for each one, as described in the following section.

4.3.1 EXPERIMENTAL METHODOLOGY

With both datasets the description learning component chosen to learn the concepts was the DL-Learner system with default settings, since it is a largely used environment to learn DLs.

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 compact terminology; and (iii) whether the ordering of concepts defined beforehand makes some difference for the terminology learning task. Thus, we run and compare DL-Learner, as the approach for single concept learning, ATL, BTL and DTL. The latter considering three cases: (1) and (2) based on the definition of $A_{\mathcal{T}}$ and avoiding cycles: (1) the order established by BTL's taxonomy, in this case $\langle GP, GF, GM \rangle$ and (2) the reversed order of the one considered in (1) ($\langle GM, GF, GP \rangle$); (3) removing cycles.

MORAL REASONER

Concerning this dataset we want 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 BTL, DTL and ATL, including a comparison with DL-Learner? We investigate this issue using quantitative and qualitative measures. For the former we considered 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 subsumption relation: (a) does the learned terminology subsumes the original one (more general)? (2) does the original terminology subsumes the learned one (more specific)? If both questions are true, then the learned and original terminology are equivalent (this would be the best result possible).

For this dataset 10-fold cross-validation was used.

4.3.2 EXPERIMENTAL RESULTS

In this section, the results obtained with the experiments are described.

KINSHIP DATASET

The results obtained considering the Kinship dataset is depicted in Table 4.4.

From that, it is possible to perceive that all concepts description follow the correspondent semantics. Moreover, BTL, DTL and ATL were able to learn a more compact and readable terminology than DL-Learner. Additionally, in order to uphold DL's assumption of acyclic terminologies we used: a version of DTL that avoids cycles ((1) and (2)), in this case orderings are particularly important, i.e., different orders entails different results; and a version of DTL that removes the cycles (3)

Table 4.4: Toy Example results: Concepts Definitions and their sizes found with the methods BTL, DTL, ATL 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
BTL	GP	EXISTS parent.EXISTS parent.TOP.	5
	GF	(grandparent AND male).	3
	GM	(grandparent AND female).	3
$DTL_{(1)}$	GP	(grandfather OR grandmother).	3
	GF	(male AND EXISTS married.grandmother).	5
	GM	(female AND EXISTS parent.EXISTS parent.TOP).	7
$DTL_{(2)}$	GP	EXISTS parent.EXISTS parent.TOP.	5
	GF	(grandparent AND male).	3
	GM	(grandparent AND female).	3
$DTL_{(3)}$	GP	EXISTS parent.EXISTS parent.TOP.	5
	GF	(grandparent AND male).	3
	GM	(grandparent AND female).	3
ATL	GP	EXISTS parent.EXISTS parent.TOP.	5
	GF	(grandparent AND male).	3
	GM	(grandparent AND female).	3

(alg. 4 and 5). The results in Table 4.4 supports such conclusion and depicted that DTL avoiding cycles could achieve the same terminology as BTL and ATL by reversing its learning order.

MORAL REASONER DATASET

In the rest of this chapter we will refer to DTL avoiding cycles as DTL_1 or $DTL1$, and to DTL removing cycles as DTL_2 or $DTL2$.

Table 4.5 exhibits the results for all the proposed approaches (BTL, DTL1, DTL2 and ATL), considering the two experiments (Exp1 and Exp2) described in Section 4.3.1 and the different percentage of training examples. The proportion of positive and negative examples was maintained in each set. Regarding BTL, we present the results for 3 different values of the threshold parameter (0.6, 0.8, 0.9). Moreover, to evaluate the approaches, we considered the percentage of original dependency among concepts found in the learned terminology (% Hits) and Table 4.5

depicts the average of the 10 folds.

Observing the results in Table 4.5, both BTL and DTL performed worse than expected. The DTL (1 and 2) behavior could be explained by it being even more sensitive to completeness than we expected, this might be the case since we direct 10% of the original examples for 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 4.6 validated our hypotheses, showing that with the whole set of individuals DTL1 was able to find all "hits" for Exp1. Furthermore, being the only approach able to achieve this. Moreover, the results for Exp2 was also improved. For Exp1 DTL2 has a comparable result to the other approaches, but it is a lot better on Exp2, this might indicate that DTL2 behaves better on more complex terminology learning problems. The results for BTL did not change as much as when the completeness was relaxed. Therefore, when all the individuals are known or when the examples are a more significant sample of the domain DTLs are the better choice. ATL's approach found far less dependencies than its counterparts, Table 4.7 shows the ATL's hits.

For the analysis of the terminology quality, DL-Learner was also considered. Fig. 4.2 and Fig. 4.3 plots the f-measure results for each of the concepts and for all scenarios. It is possible to notice that all approaches performed statistically similar in both Exp1 and Exp2, except DTL in Exp2, when inducing the definition of *weak_intend*. However, it is important to notice that in this specific case, the appropriate relationships was not considered (see Table 4.5), possibly because the lack of full completeness due to the 10% of examples reserved for validation, as discussed previously. Therefore, terminology learned with multiple concept learning approaches achieve similar predictive results compared to single concept learning approach.

Regarding the qualitative metrics, we used the whole set of examples. For the

Table 4.5: Aggregate Results of Relaxing Completeness Experiment for the methods BTL, DTL and ATL.

Approach	Experiment	Percentage	% Hits
BTL 0.9	exp1	100% (80%, 60%, 40%, 20%)	33% 0%
	exp2	100% 80% 60% 40% 20%	60% 80% 60% 40% 20%
BTL 0.8	exp1	100% (80%, 60%, 40%, 20%)	66% 0%
	exp2	100% 80% 60% 40% 20%	20% 80% 60% 40% 20%
BTL 0.6	exp1	100% 80% (60%, 40%, 20%)	66% 66% 0%
	exp2	100% 80% 60% 40% 20%	20% 20% 60% 40% 20%
DTL_1	exp1	100% (80%, 60%, 40%, 20%)	33% 0%
	exp2	(100%, 80%) 60% (40%, 20%)	20% 0% 0%
DTL_2	exp1	100% (80%, 60%, 40%, 20%)	33% 0%
	exp2	(100%, 80%) 60% (40%, 20%)	20% 0% 0%
ATL	exp1	(100%, 80%, 60%, 40%, 20%)	0%
	exp2	(100%, 80%, 60%, 40%, 20%)	0%

Table 4.6: Full Completeness over the individuals Experiment with the methods BTL, DTL and ATL.

Approach	Experiment	# Dependencies	% Hits
BTL(0.9, 0.8)	exp1	4	66%
	exp2	4	20%
BTL (0.6)	exp1	4	66%
	exp2	5	20%
DTL_1	exp1	3	100%
	exp2	6	40%
DTL_2	exp1	4	66%
	exp2	5	80%
ATL	exp1	0	0%
	exp2	0	0%

Table 4.7: Full Completeness Experiment ATL's Hits.

Coverage	Total		20%		40%		60%		80%		100%	
Fold	Exp		Exp		Exp		Exp		Exp		Exp	
	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

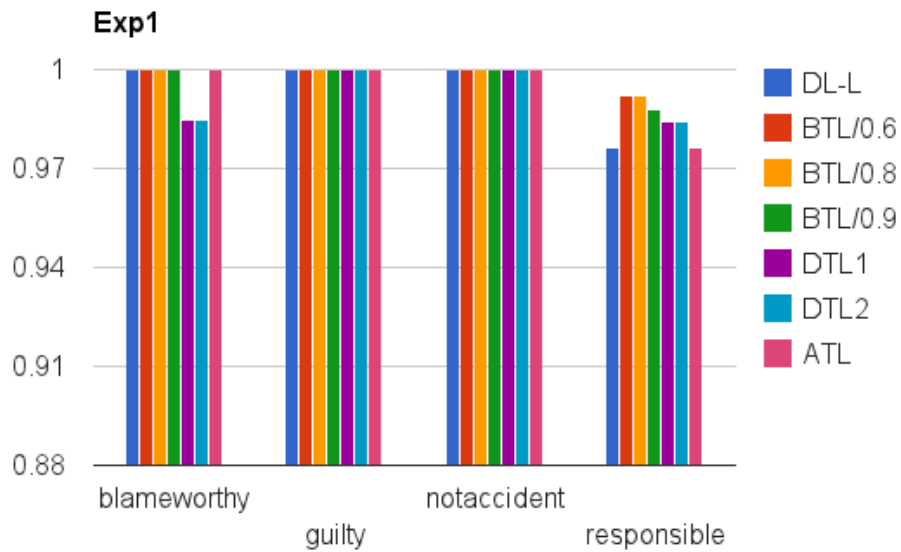


Figure 4.2: F-measure for Exp1 considering scenarios with BTL, DTL, ATL and DL-Learner (DL-L).

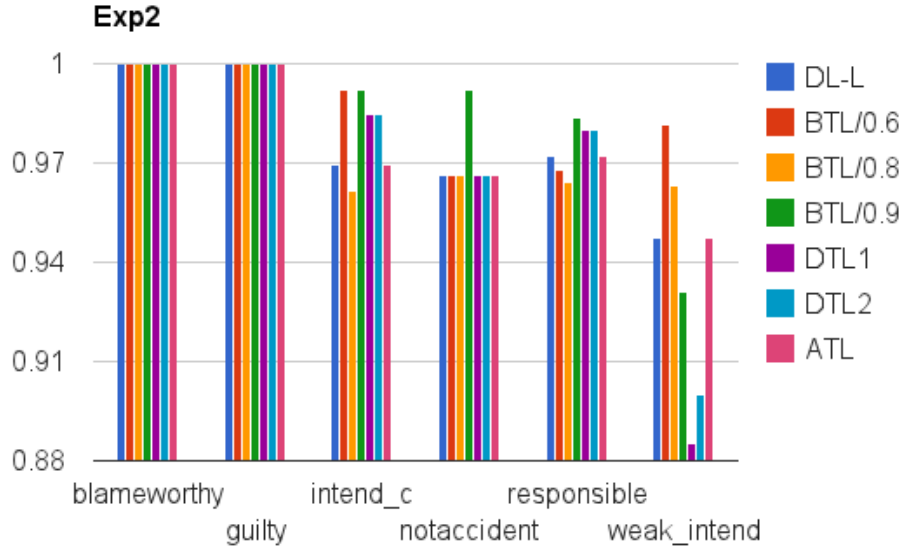


Figure 4.3: F-measure for Exp2 considering scenarios with BTL, DTL, ATL and DL-Learner (DL-L).

size of the found terminologies Fig. 4.4, on the far left we have the original size of the terminology, none of the approaches were equal or smaller than it. On the far right, we have the size for the terminologies without multiple concept learning, none of the approaches produced terminologies larger than it. ATL's results were equal to DL-Learner's because no redundancy was found. Apart from ATL, all the proposed methods induced far smaller terminologies than the equivalent concept learning without addressing the dependencies.

For the size of concept per experiment see Fig. 4.5 and Fig. 4.6, the three BTL had very similar results, being consistently better or equivalent to DL-Learner, except on "responsible" in Exp1 and "notaccident" in Exp2. DTL_1 had the best result for "guilty" in Exp1, but its "blameworthy" was the worst among our approaches, exp. Interestingly, both DTL found by far the best result for "blameworthy" and "intend_c" in Exp2, with DTL_2 being even better, while being just ok in the other results. There is a trade-off on the size of the concepts, in some instances choosing a smaller definition for a concept affects the size of other related concepts, because the latter might not be able to use the former. This explains why DTL_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.

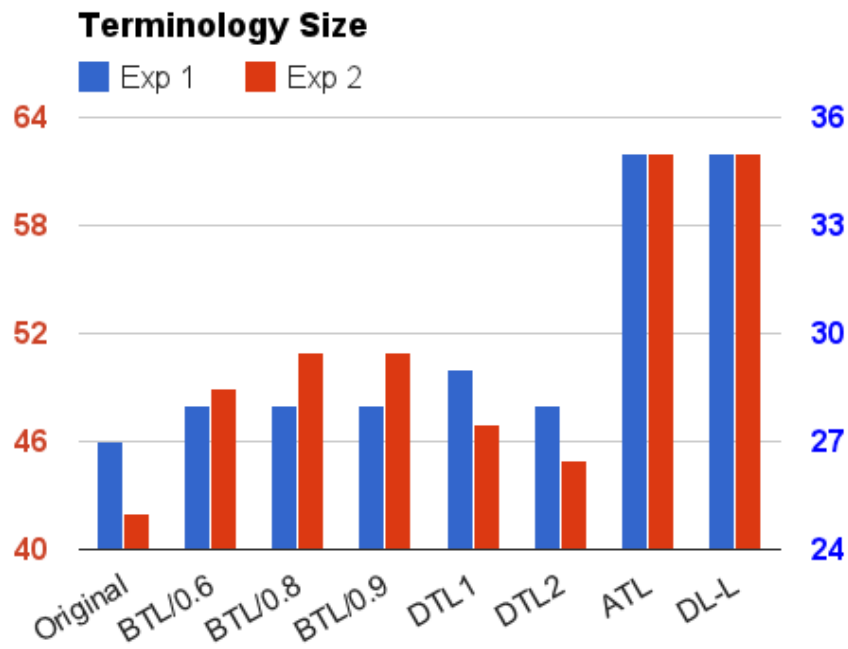


Figure 4.4: Size of found terminology for Exp1 and Exp2 with methods BTL, DTL, ATL and DL-Learner (DL-L).



Figure 4.5: Size of concepts definitions for Exp1 with methods BTL, DTL, ATL and DL-Learner (DL-L).

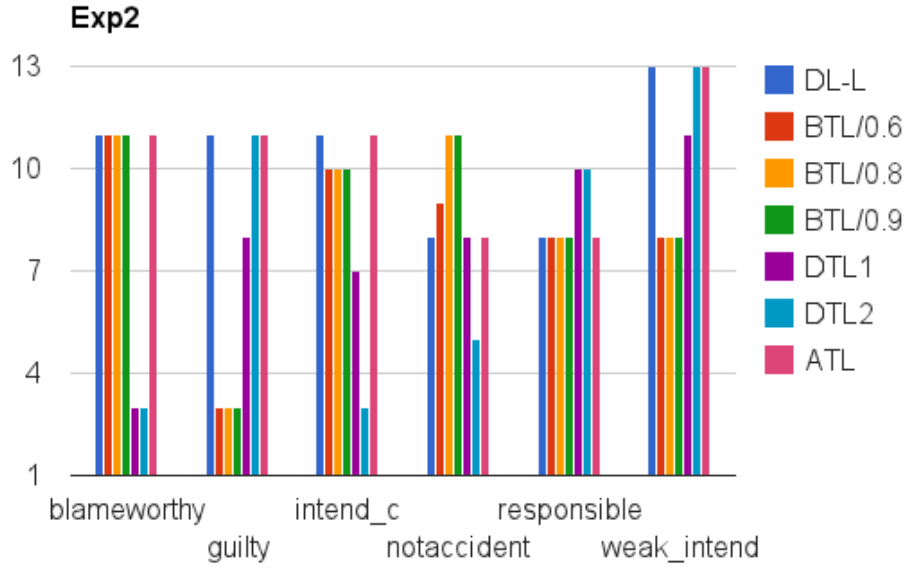


Figure 4.6: Size of concepts definitions for Exp2 with methods BTL, DTL, ATL and DL-Learner (DL-L).

Fig. 4.7 and Fig. 4.8 contrasts the size of concepts learned with their original definitions. The values are calculated through the formula $\frac{|originaldefinition|}{|learneddefinition|} - 1$, 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 DTL concerning concept "blameworthy" in Exp2. It provides the higher decrease in size. Due to the fact that DTL uses assertions about all concepts being learned and "blameworthy" is the concept that most depend on other concepts from the learning task, it is the one that most benefits from DTL approach. On the other hand, BTL will be the worst choice for this concept, since the taxonomy will be more complex.

Fig. 4.9 and Fig. 4.10 uses the same formula, showing that BTLs and DTLs provides 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 DTL are performed; (ii) definition for concepts {guilty, intend_c, weak_intend} subsumes their original definition

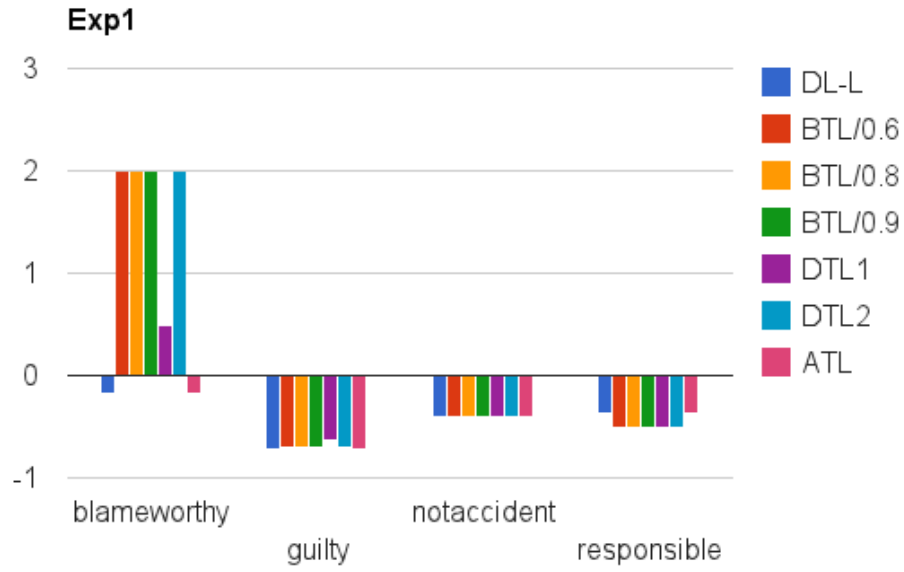


Figure 4.7: Size difference for Exp1 w.r.t. Original Definition with methods BTL, DTL, ATL and DL-Learner (DL-L).

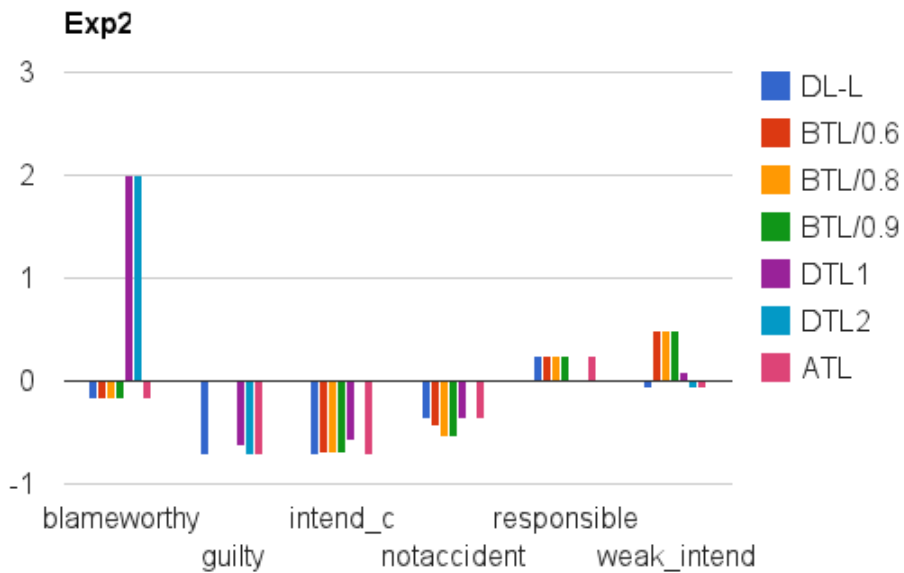


Figure 4.8: Size difference for Exp2 w.r.t. Original Definition with methods BTL, DTL, ATL and DL-Learner (DL-L).

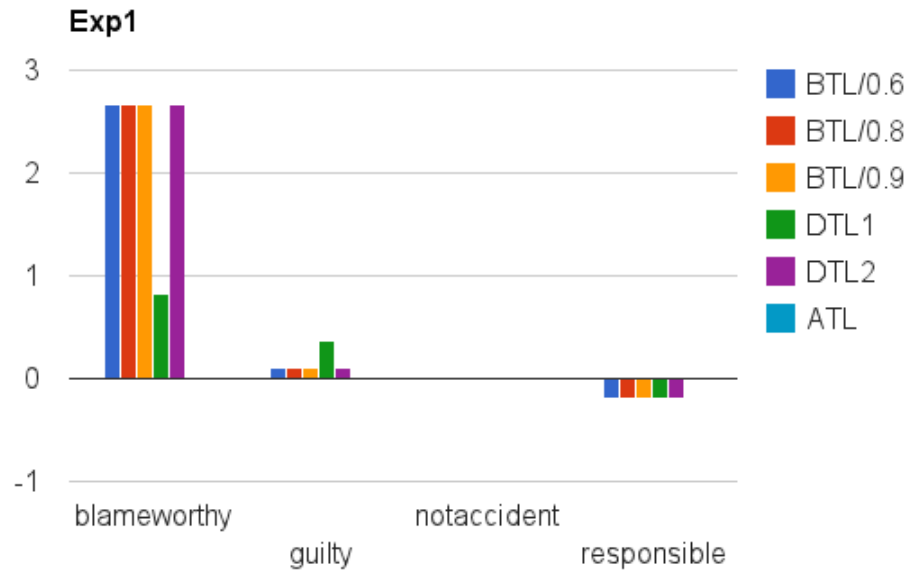


Figure 4.9: Size difference for Exp1 w.r.t. SCL with methods BTL, DTL and ATL.

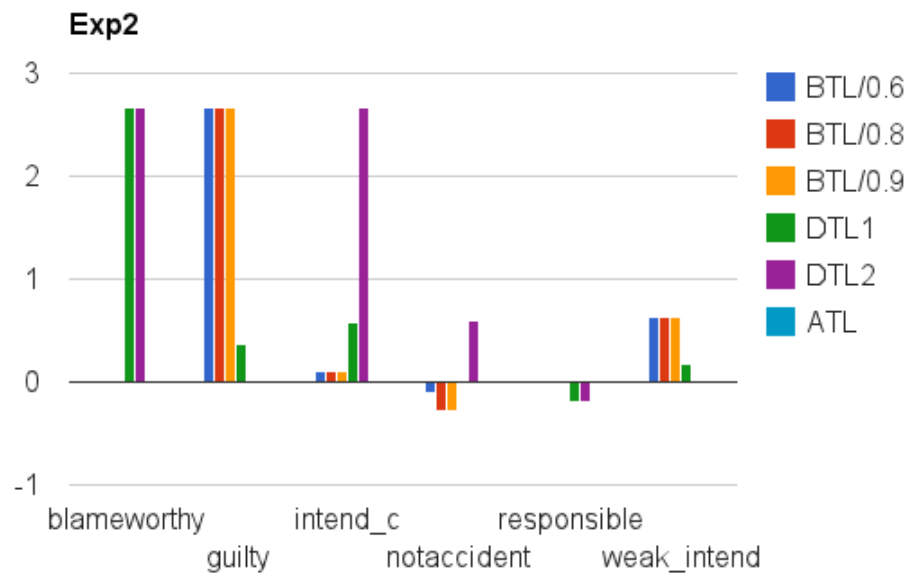


Figure 4.10: Size difference for Exp2 w.r.t. SCL with methods BTL, DTL and ATL.

when BTL is performed; (iii) the original definition for {notaccident} subsumes the definition learned with DL-Learner and (iv) the original definition for {notaccident, responsible} subsumes the learned definition when running DTL and BTL.

Summarizing, these results show the potential of the multiple concept learning approaches proposed in this work, to find smaller more readable solutions and, in some cases, solutions equivalent to those created by human beings. Furthermore, BTL proved to be less susceptible to the completeness assumption while DTL was very sensible to it. DTL_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, ATL's approach found far less dependencies than its counterparts. This is due to the fact that syntactical substitutions are far more unlikely when the set of concept learning problems have a wide range of possible solutions. This happens because the learner selects for short descriptions.

5 RELATED WORKS

There is two distinct set of related works, one regarding to the terminology learning problem (related to all three methods) and a set of related works particular to R2D2's use in theory restructuring.

5.1 TERMINOLOGY LEARNING

To the best of our knowledge, the literature related to terminology learning considering the induction of a set of related, but distinct concepts, is rather scarce. One seemingly related work is presented in (Distel, 2011), where relations among the concepts is discovery and represented through inclusion axioms (express subsumption relation). Thus it is related to BTL, where the relation among concepts is represented through a taxonomy. However, (Distel, 2011) shares a goal with (Baader et al., 2007) of finding the complete knowledge base for a DL, while we wish to deal only with a defined set of concepts.

Additionally, we can consider other works as related to ours assuming two perspectives: (i) single concept learning in DL, as such methods are used to learn the concepts definitions in all of the present methods and (ii) multiple predicates learn-

ing in ILP (Muggleton, 1991, 1992), as most of the theoretical background is shared between learning in DL and learning in ILP, and some interesting and useful ideas are bound to be shared between the two areas.

For the first case, the related works are DL-Learner (Lehmann and Hitzler, 2010), DL-FOIL (Fanizzi et al., 2008) or Yin-Yang (Iannone et al., 2007a). As discussed in Section 3, the main difference between them and our proposals is that they learn the terminology without looking into the relations existing among the concepts. Moreover, in Section 4, experiments showed that the proposed methods provide a smaller terminology compared to an approach of single concept learning.

In the second case, the most prominent related work is the one proposed in (De Raedt et al., 1993), where the problem of multiple predicate learning was defined as not limited to the generation of several independent predicate definitions but, rather involving the discovery of predicates dependencies. Although, this definition is very similar to the one defined in this paper for our multiple concept learning approaches, specific implementations are required due to particular idiosyncrasies that emerge from the particular knowledge representation - descriptions logics and logic programming, in concept learning in DL and predicate (or concept) learning in ILP, respectively. Besides the particularities of the languages, the main difference between the approaches is that, while in (De Raedt et al., 1993) a set of predicates is learned in a interleaving way, with maybe more than one definition for each predicate, this cannot be done in DL according to the upheld assumptions(*“there is only one definition for a concept name”*).

In (Malerba et al., 1997) a definition for the problem of Multiple Concept Learning is defined, a solution using statistical functions to devise the dependency level on attribute-based domains and, finally, an ad-hoc method is proposed for multiple concept learning in FOL. The motivation and definition for the Multiple Concept Learning problem removing the independence assumption are very similar to the ones used in this work. The proposed method for Multiple Concept Learning on attribute-based domains are almost identical to BTL’s solution. It creates a graph

(taxonomy) that models the dependencies among the concepts and a learning order is inferred from it, the dependencies are found using statistical methods such as χ^2 , but the idea that examples are the guides to the definitions is shared between BTL's and the method used in (Malerba et al., 1997). However, the authors states that the statistical methods are not suitable for structure domain, as is the case with FOL and DLs. Lastly, an *ad-hoc* method for Multiple Concept Learning is presented it suffers from the same problem of De Raedt et al. (1993) when transposing the ideas to the DL setting. Moreover, this method lacks any of the previous similarity with BTL's, because the graph defining the dependencies is not created within the learning process, on the contrary, it should be constructed by the user and be passed as a parameter to the method.

5.2 THEORY RESTRUCTURING - R2D2

In (Wrobel, 1996) there is a summary of works done in theory refinement for FOL. Theory refinement deals with the change of an existing theory to match a new external criteria, as oppose to learning new theories from scratch as it is done in ILP.

Because theory refinement makes changes on existing theories it can be seen as a knowledge base maintenance task. Theory refinement methods can be divided into theory revision and theory restructuring. The *revision* task changes the semantics of an existing theory in order to cope with new examples that are wrongly inferred. The *restructuring* task, on the other hand does not change the semantics of the theory, but it is intended either to increase the performance or the understanding of a theory. Figure 5.1 show a taxonomic overview of the refinement tasks.

R2D2 can be seen as a theory restructuring method for structure improving, i.e., understanding, to DLs. In this instance this proposed method might be used outside the terminology learning problem.

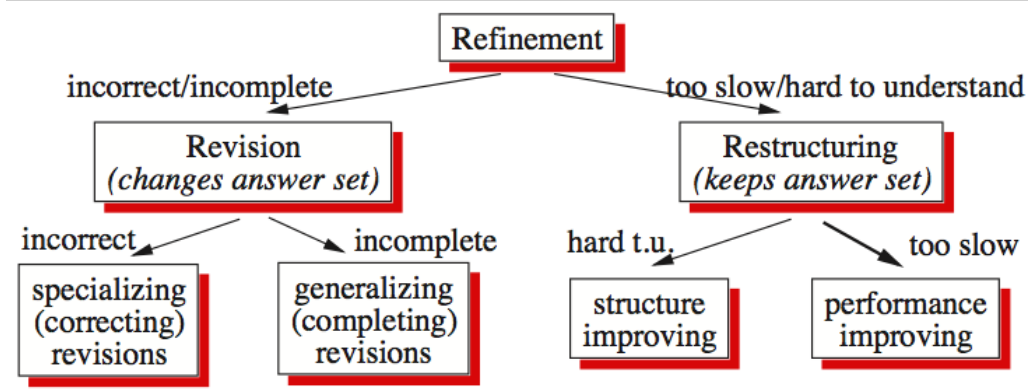


Figure 5.1: Taxonomy of theory refinement tasks (Wrobel, 1996).

Within this topic, FENDER’s (Sommer, 1995b) goal is the same as ours. FENDER is a method of *stratification* of theories, i.e., it structures a theory into layers of dependencies. It increases the number of layers, i.e., the dependencies within a theory. To do so, it uses an operator named *common partial premises*, 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 (Revoredo et al., 2006, 2007), 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 (Fonseca et al., 2003) and DLs (Calvanese, 1996), 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 commutation 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 9 presented in section 3.3.1 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 (Read and Corneil, 1977), however in special cases it has polynomial solutions: trees are in this group of special cases (Bodlaender, 1990; Faulon, 1998; Buss, 1997; Messmer and Bunke, 1996; Shamir and Tsur, 1997). We leave the investigation of more efficient tree subgraph algorithms for future works.

6 CONCLUSIONS

This research showed three strategies for terminology learning considering the induction of related, but distinct concepts in the Description Logic setting. The first one, named BTL, automatically defines an order for learning the concepts, where this order is represented by a taxonomy tree of the concepts. This is accomplished by analyzing the taxonomic intersection of the examples sets associated to each concept. The second one, named DTL, 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 take an extra care to avoid definitions with cycles or to remove them. The third method, name ATL (R2D2), is applied after the terminology learning task is concluded. It substitutes sections by a another concept, if the section is syntactically equivalent to the concept’s definition. This allows for syntactical compression while maintaining the semantics. In all the cases, we expect that the returned terminologies are compact and readable descriptions of the concepts included in the domain of discourse.

In order to validate ATL for syntactic compression we used 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.

We validated our methods by examining results yielded from two domains: a toy problem from the kinship domain and a more elaborated problem, denominated “moral reasoner”. Both of them are freely available in the DL-Learner package. In the first one, it was possible to observe that the terminology learned by BTL and DTL was more compact than the one learned with DL-Learner (standard SCL system). Moreover, DTL was more versatile than BTL. Analysing the second domain it was possible to observe that the threshold parameter for defining a proper relationship in BTL method gets more sensitive as the task is more complex. With a relatively simple task, varying this parameter did not output significantly different results. Additionally, the completeness of individuals interferes in both BTL and DTL, the later being more sensible, i.e. varying the size of the training set, and as a consequence, the representativeness of the original individuals in the examples set of each concept, yield different quality of terminology. 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 ATL, were able to fetch more compact and readable terminologies than the SCL. Both DTL’s methods outperformed all the other methods, especially DTL_2 on Exp2. However, this improved performance comes at a price, DTL_2 has to learn every alternative

definitions in order to remove existing cycles. ATL's approach found far less dependencies than its counterparts. This is due to the fact that syntactical substitutions are far more unlikely when the set of concept learning problems have a wide range of possible solutions. This happens because the learner selects for short descriptions.

6.1 CONTRIBUTIONS

The major contributions of this research are:

- The empirical result that removing the independence assumption smaller terminologies can be induced;
- Before Terminology Learning method;
- During Terminology Learning method;
 - With cycle avoidance;
 - With cycle removal;
- After Terminology Learning (R2D2) method;

Some of the minor contributions are:

- Argumentation and definitions related to:
 - Multiple Concept Learning in Description Logics;
 - Simplification;
 - Substitution of a concept definition for its sets of examples;
- A tree representation for concept's descriptions that deals with commutable constructors;

6.2 FUTURE WORKS

An integral part of the terminology learning task involves learning a single concept definition, which is learned using one of the existing methods. As future work, a study of the underlying bias of this part of the process will be done, because each method has a slightly different focus, either on the general to specific direction (Lehmann and Hitzler, 2010; Fanizzi et al., 2008) or in the specific to general (Iannone et al., 2007b) that may yields different results.

For ATL: i) Alternative more efficient algorithms for ATL tree isomorphism; ii) Allow the algorithms to find groups that are not a full concept definition, this will allow the generation of new concepts, and a greater syntactical compression.

Investigate methods to remove deeper cycles for DTL with cycle removal.

Further experiments on different domains are necessary. Also, a interesting questions is how does the methods behave in datasets with noise.

On this research we intuitively assumed that the methods are language independent, in a future work other DL languages and probabilistic DLs (Revoredo et al., 2011) should be tested.

In future work, we would like to investigate if there is any gain in using this method in a terminology learning process (Melo et al., 2013b,a) and also to investigate alternative more efficient tree isomorphism algorithms.



BIBLIOGRAPHY

- Anderson, J. R., Michalski, R. S., Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (1986). *Machine learning: An artificial intelligence approach*, volume 2. Morgan Kaufmann.
- Baader, F., Ganter, B., Sertkaya, B., and Sattler, U. (2007). Completing description logic knowledge bases using formal concept analysis. In *IJCAI*, pages 230–235.
- Baader, F. and Nutt, W. (2010). Basic description logics. In Baader, F., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors, *The description logic handbook*, pages 47–100. Cambridge University Press, 2 edition.
- Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The semantic web. *Scientific American*, 5(284):34.
- Bodlaender, H. L. (1990). Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631 – 643.
- Brachman, R. J. and Levesque, H. J. (2004). *Knowledge Representation and Reasoning*. Elsevier, 1st edition edition.
- Buss, S. (1997). Alogtime algorithms for tree isomorphism, comparison, and canonization. In Gottlob, G., Leitsch, A., and Mundici, D., editors, *Computational*

- Logic and Proof Theory*, volume 1289 of *Lecture Notes in Computer Science*, pages 18–33. Springer Berlin Heidelberg.
- Calvanese, D. (1996). Finite model reasoning in description logics. *KR*, 96:292–303.
- De Raedt, L., Lavrac, N., and Dzeroski, S. (1993). Multiple predicate learning. In *Proceedings of the 13th international joint conference on artificial intelligence*, pages 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*, pages 37–43.
- Dowling, W. F. and Gallier, J. H. (1984). Linear-time algorithms for testing the satisfiability of propositional horn formulae. *The Journal of Logic Programming*, 1(3):267–284.
- Fanizzi, N., d’Amato, C., and Esposito, F. (2008). DL-foil concept learning in description logics. In *Proceedings of the 18th International Conference on Inductive Logic Programming (ILP-2008)*, volume 5194 LNAI of *Lecture Notes in Computer Science*, pages 107–121. Springer.
- Faulon, J.-L. (1998). 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.
- Fonseca, N., Rocha, R., Camacho, R., and Silva, F. (2003). Efficient data structures for inductive logic programming. In *Inductive Logic Programming*, pages 130–145. Springer.
- Horn, A. (1951). On sentences which are true of direct unions of algebras. *Journal of symbolic logic*, pages 14–21.

- Hume, D. (2000). *An enquiry concerning human understanding: A critical edition*, volume 3. Oxford University Press.
- Huth, M. and Ryan, M. (2004). *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge University Press.
- Iannone, L., Palmisano, I., and Fanizzi, N. (2007a). An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, 26(2):139–159.
- Iannone, L., Palmisano, I., and Fanizzi, N. (2007b). An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, 26(2):139–159.
- Lavrac, N. and Dzeroski, S. (1994). Inductive logic programming. In *WLP*, pages 146–160. Springer.
- Lehmann, J. (2007). Hybrid learning of ontology classes. In *Machine Learning and Data Mining in Pattern Recognition*, pages 883–898. Springer.
- Lehmann, J. (2009). DL-learner: Learning concepts in description logics. *Journal of Machine Learning Research*, 10:2639–2642.
- Lehmann, J. and Hitzler, P. (2008a). *Foundations of refinement operators for description logics*, volume 4894 LNAI of *Lecture Notes in Computer Science*. Springer.
- Lehmann, J. and Hitzler, P. (2008b). A refinement operator based learning algorithm for the $\{ALC\}$ description logic. In *Inductive Logic Programming*, pages 147–160. Springer.
- Lehmann, J. and Hitzler, P. (2010). Concept learning in description logics using refinement operators. *Machine Learning*, 78(1-2):203–250.
- Maedche, A. and Staab, S. (2001). Ontology learning for the semantic web. *IEEE Intelligent Systems and Their Applications*, 16(2):72–79.

- Malerba, D., Semeraro, G., and Esposito, F. (1997). A multistrategy approach to learning multiple dependent concepts. *Machine learning and statistics: The interface*, pages 87–106.
- Melo, R., Revoredo, K., and Paes, A. (2013a). Learning multiple description logics concepts. In *ILP 2013 Late Breaking Papers "to appear"*.
- Melo, R., Revoredo, K., and Paes, A. (2013b). Terminology learning through taxonomy discovery. In *2013 Brazilian Conference on Intelligent Systems*, pages 169–174.
- Melo, R., Revoredo, K., and Paes, A. (2014). Syntactic compression of description logics terminologies. In *Brazilian Conference on Intelligent Systems "submitted"*.
- Messmer, B. and Bunke, H. (1996). Subgraph isomorphism detection in polynomial time on preprocessed model graphs. In Li, S., Mital, D., Teoh, E., and Wang, H., editors, *Recent Developments in Computer Vision*, volume 1035 of *Lecture Notes in Computer Science*, pages 373–382. Springer Berlin Heidelberg.
- 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.
- Muggleton, S. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679.
- Quinlan, J. R. and Cameron-Jones, R. M. (1993). Foil: A midterm report. In *Machine Learning: ECML-93*, pages 1–20. Springer.
- Read, R. C. and Corneil, D. G. (1977). The graph isomorphism disease. *Journal of Graph Theory*, 1(4):339–363.

- Revoredo, K., Ochoa-Luna, J., and Cozman, F. (2011). Learning probabilistic description logics: A framework and algorithms. In *In proceedings of the MICAI*, volume 7094 of *LNCS*, pages 28–39. Springer.
- Revoredo, K., Paes, A., Zaverucha, G., and Costa, V. S. (2006). 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.
- Revoredo, K., Paes, A., Zaverucha, G., and Costa, V. S. (2007). Combinando invenção de predicados e revisão de teorias de primeira-ordem probabilísticas. In *VI ENIA*.
- Robinson, J. A. (1965). A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41.
- Schmidt-Schauss, M. and Smolka, G. (1991). Attributive concept descriptions with complements. *Artificial Intelligence*, 48:1–26.
- Shamir, R. and Tsur, D. (1997). Faster subtree isomorphism. In *Theory of Computing and Systems, 1997., Proceedings of the Fifth Israeli Symposium on*, pages 126–131.
- Smolensky, P. (1987). Connectionist ai, symbolic ai, and the brain. *Artificial Intelligence Review*, 1(2):95–109.
- Sommer, E. (1995a). An approach to quantifying the quality of induced theories. In *Proceedings of the IJCAI 95-Workshop on Machine Learning and Comprehensibility*. Citeseer.
- Sommer, E. (1995b). Fender: An approach to theory restructuring. In *Machine Learning: ECML-95*, pages 356–359. Springer.
- Wogulis, J. L. (1994). *An approach to repairing and evaluating first-order theories*

containing multiple concepts and negation. PhD thesis, University of California at Irvine.

Wrobel, S. (1996). First order theory refinement. *Advances in inductive logic programming*, 32:14–33.

Appendix A

OWA AND CWA AND THEIR RELATIONSHIP WITH INFERENTIAL AND INDUCTIVE REASONING

Usually *Closed World Assumption (CWA)* and *Open World Assumption (OWA)* are related with inferential knowledge (deductive knowledge), in this case:

Definition 10 (Closed World Assumption). *The truth-value of statements not currently known to be true is false.*

Definition 11 (Open World Assumption). *The truth-value of a statement is independent of whether or not it is currently known.*

When CWA is upheld, from Definition 10, an answer for a query can only be “True” or “False”, i.e., and statement is either a logical consequence or not (Example 10). Otherwise, when OWA is upheld, from Definition 11, an answer for a query can be “True”, “False” or “Unknown”, something is unknown when it: i) does not contradict any of the conditions and ii) does not satisfy all the necessary conditions (Example 11).

Example 10. *Using the kinship domain with its default semantics. Have a individual $m1$ whose only known assertion is “male($m1$)”; and individual $f1$, whose only known assertion is “female($f1$)”. Let the concept definition for mother be: $mother \equiv female \sqcap \exists hasChild.\top$.*

When upholding CWA, it would produce the following answers:

mother($f1$)? False. mother($m1$)? False.

Example 11. *Using the kinship domain with its default semantics. Have a individual $m1$ whose only known assertion is “male($m1$)”; and individual $f1$, whose only known assertion is “female($f1$)”. Let the concept definition for mother be: $mother \equiv female \sqcap \exists hasChild.\top$.*

When upholding OWA, it would produce the following answers:

mother($f1$)? Unknown. mother($m1$)? False.

In the context of this work, we are dealing with inductive learning methods. In the inductive setting, one may see a connection between the closed or open assumption and the *Problem of induction*. The original problem of induction can be simply put. It concerns the support or justification of inductive methods. Inductive methods lack justification for either:

1. Generalizing the properties of a class of objects based on a finite number of observations of particular instances of a class (for example, the inference that “all swans seen so far are white, therefore all swans are white”, before a black swan is known) or
2. The uniformity of nature(Hume, 2000)

The CWA and OWA in the case of inductive learning are related to point 1, whether or not a inducted theory/terminology is justified based on the available examples. It has been argued in the past that such problem is unsolvable (Hume, 2000), i.e., there is no logical justification for generalization based on a finite set of examples (OWA). In practice this means that a learned theory/terminology should

be only tentatively accepted while no new contradictory examples are presented, i.e., a theory is only correct with regards to a specific domain (CWA). Therefore, while using induction to learn a theory/terminology one is in practice using a type of CWA, in the sense that the induced theory/terminology is “only valid” on the learning domain, one cannot assert that a finite domain is representative of the universe.

Point 1 is also the reason behind the need of methods for theory revision.

One could also view a CWA and OWA with regards to the sets of examples for the induction of a theory/terminology. When learning a theory \mathcal{T}_i , using a background knowledge \mathcal{KB} and a set of examples \mathcal{E}_i , on the learning domain \mathcal{D}_l ; \mathcal{E}_i can be divided into:

- **Positives:** examples that should be covered by \mathcal{T}_i ;
- **Negatives:** examples that should not be covered by \mathcal{T}_i ;
- **Ignored:** examples that should not directly affect the inductive learning, but can either belong or not to \mathcal{T}_i .

If, for a domain \mathcal{D}_l , it is possible to enumerate all positive examples \mathcal{E}_{i_p} , the set of negative examples \mathcal{E}_{i_n} is the complement of \mathcal{E}_{i_p} w.r.t. \mathcal{D}_l . This is the case, because within the inductive learning method a individual either should be covered by the candidate hypothesis or it should not, if all the individuals that should be covered by \mathcal{T}_i are known all the others does not satisfy the requirements (w.r.t. \mathcal{D}_l) of \mathcal{T}_i . When this restriction is meet, this can be seen as under the CWA¹.

¹On BTL and DTL we are referring to this type of CWA.