Discovering multi-relational association rules from ontological knowledge bases to enrich ontologies

Duc Minh Tran

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Découverte de règles d'association multi-relationnelles à partir de bases de connaissances ontologiques pour l'enrichissement d'ontologies

Duc Minh TRAN

Inria-I3s

Présentée en vue de l'obtention du grade de docteur en Informatique
d'Université Côte d'Azur
et d'Université Da Nang
Dirigée par : *Andrea G.B. TETTAMANZI*
Co-dirigée par : *Thanh Binh NGUYEN*
Soutenue le : 23-07-2018

Après avis de :
Elena MARCHIORI, Professeur, Université Radboud
Huu Hanh HOANG, Professeur associé, Université Hue
Marie-Laure MUGNIER, Professeur, Université Montpellier

Devant le jury, composé de :
Thanh Nhan LE, Professeur, Université Côte d'Azur
Andrea G.B. TETTAMANZI, Professeur, Université Côte d'Azur
Thanh Binh NGUYEN, Professeur associé, Université Da Nang
Marie-Laure MUGNIER, Professeur, Université Montpellier
Huu Hanh HOANG, Professeur associé, Université Hue

i3s sophia antipolis

Inria

Erasmus Mundus

Acknowledgement

First and foremost, I have to thank my research advisors, Professor Andrea G.B. Tettamanzi and Professor NGUYEN Thanh Binh. Without their assistance and dedicated involvement in every step throughout the process, this thesis would have never been accomplished. I would like to thank you very much for your support and understanding over these past four years.

Besides my advisors, I would like to thank the rest of my thesis committee: Professor Marie-Laure MUGNIER, Professor HOANG Huu Hanh, and Professor Elena MARCHIORI, for their insightful comments and encouragement, but also for the hard question which incented me to widen my research from various perspectives.

I am grateful to Professor LE Thanh Nhan for having accepted to be the examiner of this thesis. I also would like to thank Olivier CORBY who gave me meaningful comments to improve the presentation.

My sincere thanks also goes to Claudia D'Amato. Without her precious support, I would not be difficult to reach the results this thesis. I cannot adequately express how thankful I am.

I would like to thank to my fellow lab-mates in Wimmics team for giving me a friendly atmosphere. I will never forget the amazing things that I saw and the experiences here.

I would like to thank the I3S laboratory and INRIA for having welcomed me and creating an ideal working environment for me.

For the EMMA Scholarship Fund, Erasmus Plus Program and DNIIT (Danang International Institute of Technology), I cannot thank you enough for giving me the financial support which gave me a chance to study and learn in France.

Last but not the least, I would like to thank my family: my parents, my wife, my daughters and to my brothers and sisters for supporting me spiritually throughout writing this thesis and my life in general.

Découverte de règles d'association multi-relationnelles à partir de bases de connaissances ontologiques pour l'enrichissement d'ontologies

Résumé: Dans le contexte du Web sémantique, les ontologies OWL représentent des connaissance explicite sur un domaine sur la base d'une conceptualisation des domaines d'intérêt, tandis que la connaissance correspondante sur les individus est donnée par les données RDF qui s'y réfèrent. Dans cette thèse, sur la base d'idées dérivées de l'ILP, nous visons à découvrir des motifs de connaissance cachés sous la forme de règles d'association multi-relationnelles en exploitant l'évidence provenant des assertions contenues dans les bases de connaissances ontologiques. Plus précisément, les règles découvertes sont codées en SWRL pour être facilement intégrées dans l'ontologie, enrichissant ainsi son pouvoir expressif et augmentant les connaissances sur les individus (assertions) qui en peuvent être dérivées. Deux algorithmes appliqués aux bases de connaissances ontologiques peuplées sont proposés pour trouver des règles à forte puissance inductive: (i) un algorithme de génération et test par niveaux et (ii) un algorithme évolutif. Nous avons effectué des expériences sur des ontologies accessibles au public, validant les performances de notre approche et les comparant avec les principaux systèmes de l'état de l'art. En outre, nous effectuons une comparaison des métriques asymétriques les plus repandues, proposées à l'origine pour la notation de règles d'association, comme éléments constitutifs d'une fonction de fitness pour l'algorithme évolutif afin de sélectionner les métriques qui conviennent à la sémantique des données. Afin d'améliorer les performances du système, nous avons proposé de construire un algorithme pour calculer les métriques au lieu d'interroger via SPARQL-DL.

Mots clés : Web sémantique, Ontologie, OWL, SWRL, RDF, Exploration de données, Algorithmes évolutionnaires, Logique de description, Découverte de modèle

Discovering multi-relational association rules from ontological knowledge bases to enrich ontologies

Abstract :

In the Semantic Web context, OWL ontologies represent explicit domain knowledge based on the conceptualization of domains of interest while the corresponding assertional knowledge is given by RDF data referring to them. In this thesis, based on ideas derived from ILP, we aim at discovering hidden knowledge patterns in the form of multi-relational association rules by exploiting the evidence coming from the assertional data of ontological knowledge bases. Specifically, discovered rules are coded in SWRL to easily integrated within the ontology, thus enriching its expressive power and augmenting the assertional knowledge that can be derived. Two algorithms applied to populated ontological knowledge bases are proposed for finding rules with a high inductive power: (i) level-wise generated-and-test algorithm and (ii) evolutionary algorithm. We performed experiments on publicly available ontologies, validating the performances of our approach and comparing them with the main state-of-the-art systems. In addition, we carry out a comparison of popular asymmetric metrics, originally proposed for scoring association rules, as building blocks for a fitness function for evolutionary algorithm to select metrics that are suitable with data semantics. In order to improve the system performance, we proposed to build an algorithm to compute metrics instead of querying via SPARQL-DL.

Keywords: Semantic Web, Ontology, OWL, SWRL, RDF, Data mining, Evolutionary Algorithms, Description Logics, Pattern Discovery.

Table of Contents

1 Introduction **1**

1.1 Introduction and objectives 1

1.1.1 Introduction 1

1.1.2 Objectives 3

1.2 Research questions 4

1.3 Research Methodology 4

1.4 Contributions of the thesis 6

1.5 Publications 9

1.6 Thesis outline 9

2 State of the Art Review **11**

2.1 Semantic Web Mining 11

2.2 Expert Rule Mining 12

2.3 Ontology learning 12

2.3.1 Concept learning 12

2.3.2 Pattern mining 13

2.3.3 Evolutionary algorithm for learning 14

2.4 Pattern evaluation 14

3 Background **17**

3.1 Introduction 17

3.2 Inductive Logic Programming 18

3.3 Description Logics 19

3.3.1 Knowledge Base 19

3.3.2 Semantics 22

3.3.3 Web Ontology Language OWL 24

3.4 Semantic Web Rule Language (SWRL) 28

3.5 Language Bias 29

3.6 Basic metrics for rules evaluation 32

3.7 Extensive metrics for rules evaluation 35

3.8 Evaluation of Rule Precision 37

4 Level-Wise Generate-And-Test **39**
4.1 Introduction . . . 39
4.2 Downward Refinement Operator . . . 41
4.2.1 Preliminaries . . . 41
4.2.2 Definition of the downward refinement operator . . . 42
4.3 The algorithm . . . 43
4.3.1 Discover all possible frequent patterns . . . 44
4.3.2 Obtain multi-relational association rules . . . 55
4.4 Experiments and evaluation . . . 55
4.4.1 Experimental protocol . . . 55
4.4.2 Experimental evaluation . . . 59
4.5 Summary . . . 62

5 Genetic Algorithm **63**
5.1 Introduction . . . 63
5.2 Genetic Algorithm . . . 66
5.2.1 Introduction . . . 66
5.2.2 Terminology . . . 67
5.2.3 Structure . . . 67
5.2.4 Steady-state genetic algorithm . . . 68
5.3 The algorithm . . . 69
5.3.1 Representation . . . 69
5.3.2 Operators . . . 72
5.3.3 Fitness function . . . 79
5.3.4 Consistency check . . . 79
5.3.5 Language bias . . . 80
5.4 Experiments and evaluation . . . 80
5.4.1 The ability to predict assertional knowledge . . . 82
5.4.2 The importance and the value added . . . 83
5.5 Summary . . . 86

6 Comparing Rule Evaluation Metrics **87**
6.1 Introduction . . . 87
6.2 Experiment and evaluation . . . 88

6.2.1 Compare the number of rules discovered and the ability to predict assertional knowledge . . . 89
6.2.2 The importance and the value added . . . 91
6.3 Summary . . . 92

7 Query Optimization **99**
7.1 Introduction . . . 99
7.2 From SWRL rule to SPARQL-DL Query . . . 101
7.2.1 SPARQL-DL Query . . . 101
7.2.2 Calculate metrics with SPARQL-DL . . . 101
7.3 The algorithm . . . 103
7.3.1 Representation . . . 104
7.3.2 The individual matrix "Add a concept atom" . . . 106
7.3.3 The individual matrix "Add a role atom" . . . 108
7.4 How to compute metrics with the query optimization algorithm . . . 119
7.4.1 Support . . . 120
7.4.2 Head Coverage . . . 121
7.4.3 Confidence . . . 122
7.4.4 PCA-Confidence . . . 123
7.5 Evaluation . . . 123

8 Conclusions and perspectives **127**
8.1 Conclusions . . . 127
8.1.1 Discovering hidden knowledge patterns . . . 128
8.1.2 Selecting the best asymmetric metrics . . . 129
8.1.3 Improving computing speed . . . 130
8.2 Future Work . . . 131

A The algorithm for generating a closed rule **133**
A.1 An algorithm description . . . 133
A.1.1 The adjusted atom is a concept atom . . . 133
A.1.2 The adjusted atom is a role atom . . . 139
A.1.3 Pattern adjustment . . . 150

Bibliography **151**

List of Figures

1.1 Linking Open Data cloud diagram 2017 [AAC] 2
1.2 RDF Graph 3
1.3 A schematic illustration of the research methodology 5

3.1 OWL DL descriptions, data ranges, properties, individuals and data values syntax and semantics [Obi] 26
3.2 OWL DL axioms and facts [Obi] 27

4.1 The space of rules 40
4.2 Overview of the algorithm 41
4.3 The space of rules 58

5.1 The search space of rules used in genetic algorithm 64
5.2 Overview of the genetic algorithm 65
5.3 Structure of the genetic algorithm 68
5.4 Selection operator for crossover 73
5.5 The growth of population over generations 81

7.1 Selection operator for crossover 101
7.2 Execution time of the Level-wise Generate-And-Test algorithm . . . 124
7.3 Execution time of the Evolutionary algorithm 125

List of Tables

3.1 Syntax and semantics of $\mathcal{SROIQ}$ constructors 23
3.2 Syntax and semantics of $\mathcal{SROIQ}$ axioms 24
3.3 A rule and a KB with assertions about two relations between people and pets. 34

4.1 Key facts about the ontological KBs used. 55
4.2 Average performance metrics on each ontology 59
4.3 Comparison # extracted rules: AMIE vs LW-GAT. 60

5.1 Averageg (± st.dev.) performance on each ontology ($f_H(r) = HeadCoverage(r)$) 82
5.2 Avg (± st.dev.) performance on each ontology ($f_{HPCA}(r) = HeadCoverage(r) + PCAConfidence(r)$) 83
5.3 Comparison of the number of discovered rules. 84
5.4 Comparison of the number of extracted predictions. 85

6.1 Symbols and range of metrics (the effective range is used to assist in the choice of θ_{fit}.) 89
6.2 Comparison of the metrics by the number of discovered rules. 90
6.3 Avg (± st.dev.) performance on each ontology of HeadCoverage (H), Confidence (C) 93
6.4 Avg (± st.dev.) performance on each ontology of PCA-Confidence (P), Laplace(L) 94
6.5 Avg (± st.dev.) performance on each ontology of Conviction(CV), Certainty Factor(CF) 95
6.6 Avg (± st.dev.) performance on each ontology of Added value (A), J-Measure (J) 96
6.7 Avg (± st.dev.) performance on each ontology of Gini factor (G) . . 97
6.8 Comparison of the number of discovered rules. 98

Acronyme

CWA Closed World Assumption

DL Description logic

EA Evolutionary Algorithm

FOIL First-order inductive learner

FOL First-order logic

ILP Inductive Logic Programming

LOD Linked Open Data

OWA Open World Assumption

OWL Web Ontology Language

PCA Partial Completeness Assumption

RDF Resource Description Framework

RDFS Resource Description Framework Schema

RDBMS Relational Database Management System

RuleML Rule Markup Language

SPARQL SPARQL Protocol and RDF Query Language

SPARQL-DL SPARQL Query for OWL-DL

SW Semantic Web

SWRL Semantic Web Rule Language

W3C World Wide Web Consortium

Chapter 1

Introduction

Contents

1.1 Introduction and objectives 1
1.1.1 Introduction 1
1.1.2 Objectives 3
1.2 Research questions 4
1.3 Research Methodology 4
1.4 Contributions of the thesis 6
1.5 Publications 9
1.6 Thesis outline 9

1.1 Introduction and objectives

1.1.1 Introduction

The Semantic Web has undergone a steady and continuous development toward its primary objective, which is to associate meaning with the data and to exploit the data through intelligent processing techniques. On the Semantic Web, data must be mapped to the RDF, which is a data model [KC04] of the data layer of the Semantic Web. This data model is designed for the integrated representation of information obtained from multiple sources and creates new structured information from information in unstructured form. In RDF, information is represented in RDF triples [KC04] which allows us to define statements about resources in the form of subject-predicate-object expressions.

Linked Data, also known as the Web of Data, is one of the core concepts and pillars of the Semantic Web. This is a way of publishing structured data linked to each other in order to make it easy to query semantics. We use Linked Data

to share data in a way that computers can automatically read information instead of for human readers. Linked Open Data (LOD) is Linked Data which is released under an open license, which does not impede its reuse for free. Up to now, there are 1,163 datasets and billions of RDF triples are available on the LOD Cloud (Figure 1.1). In order to retrieve information from LOD, we use a query language called SPARQL (through a number of SPARQL endpoints) to perform semantic queries. These semantic queries enable the retrieval of both explicitly and implicitly derived information based on the syntax of SPARQL.

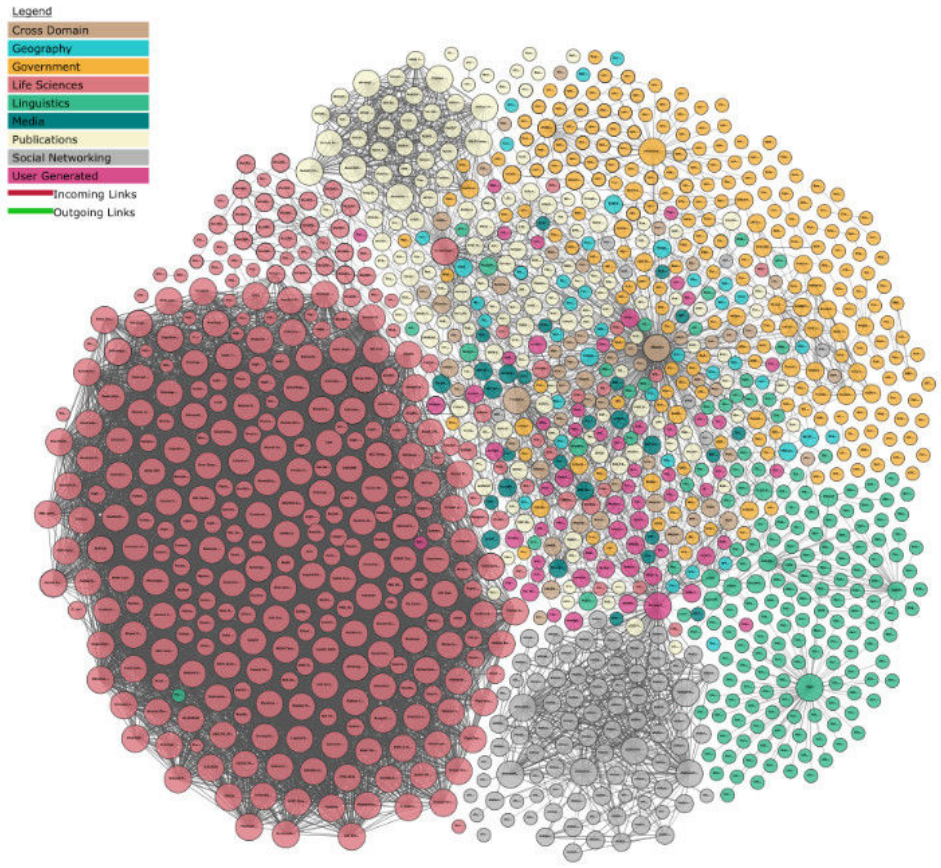


Figure 1.1: Linking Open Data cloud diagram 2017 [AAC]

LOD could be considered as a huge knowledge base containing a large number of facts in many different fields, and what motivated the research that we have undertaken in this thesis is the opportunity of analyzing wealth of data that are available on the LOD with the aim of extracting information from it, with respect to a specific domain of discourse, i.e., learn knowledge from it.

By combining RDF triples, we obtain an RDF Graph (Fingure 1.2). Therefore, general methods and techniques that have been employed for mining graphs should be relevant to mining LOD.

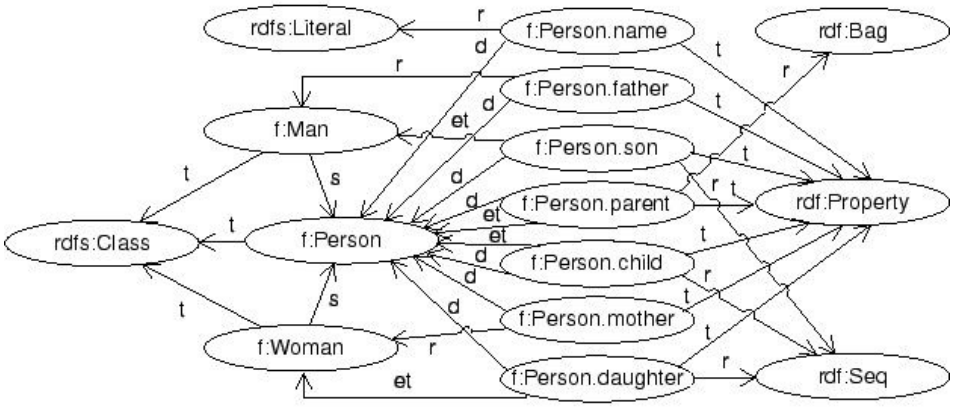


Figure 1.2: RDF Graph

1.1.2 Objectives

1.1.2.1 Overall objective

As noted above (Section 1.1.1), LOD contains a huge amount of data in many different areas, therefore mining the data extracted from LOD is an obvious demand. The overall objective of the thesis is to build an algorithm which is used for the purpose of inductive learning from assertional data in the ontology. The kind of learned knowledge will be reintegrated into the ontology to enrich the data. Experimental evaluation is performed on publicly available ontologies.

1.1.2.2 Specific objectives

The specific objectives of the thesis, which are a consequence of the overal objective stored in Section 1.1.2.1 are the following:

- First, we must determine the kind of knowledge to be learned, so that after being extracted, that kind of knowledge will be tested using deductive process.
- Select solutions for the algorithm and define search operators along with biases to avoid discovering inconsistent, redundant, and trivial knowledge, thereby

minimizing search efforts.

- Select or propose the best metrics to evaluate the quality of learned knowledge. We are particularly interested in the metrics that are tailored to comply with the open world assumption (OWA) scenario.

- Select or propose tools or metrics to assess the accuracy of learned knowledge.

- Provide tools for automatically extracting (learning) knowledge from the ontology through a specific application. Users can use the output results of this application immediately.

- The research results of the thesis will be applied to publicly available ontologies and compared to the best state-of-the-art systems.

1.2 Research questions

Based on the above research objectives, we have developed the following research questions:

- **Research question 1 :**
  What kind of knowledge is learned from the ontology?

- **Research question 2 :**
  What mining methods are applied to extract (to learn) knowledge from the ontology and which mining method is optimal ?

- **Research question 3 :**
  What methods of evaluation are used to assess the kind of knowledge learned from the ontology and which method of evaluation is the best ?

1.3 Research Methodology

Figure 1.3 illustrates our research methodology. Firstly, we determine a kind of knowledge that is extracted or learned from knowledge base. Specifically, this kind of knowledge is expressed in the form of rules standardized by the W3C and it can be easily integrated in the ontology. Next, we automatically generate these rules by

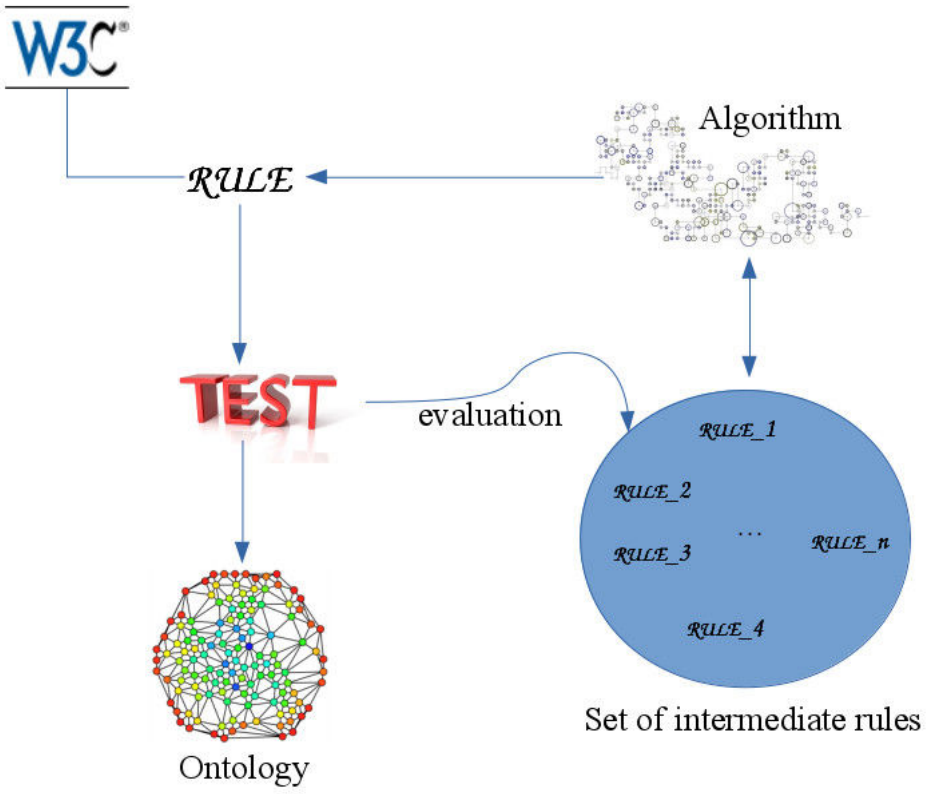


Figure 1.3: A schematic illustration of the research methodology

algorithms based on ideas derived from Inductive Logic Programming (ILP) which is a method for inductively learning rules from facts and the generation of these rules lays outside of logic (meaning based on frequently occurring events without deductive process). These algorithms should be designed to be able to reach results in an acceptable execution time: After that, generated rules will be tested by some tools such as metrics, a deductive process. A stop condition of the algorithm is met when one of the following conditions is satisfied: (i) The algorithm has traversed the whole search space of rules, in case we are dealing with a small search space. (ii) Otherwise, if we are dealing with a huge search space of rules, some constraint conditions of the algorithm are created in order to try to retrieve the best rules in the largest quantity. During the execution of the algorithm, if the stopping condition of the algorithm has not occurred, the rules can be added to a set of intermediate rules, that can be combined by the algorithm to generate new rules. After the stopping condition of the algorithm is reached, the extracted rules (known

as discovered rules) will be integrated into the ontology to enrich it.

In practice, the rule space might be too large to be exhaustively searched, and the data mining algorithm will have to use some heuristic methods or soft computing techniques. In particular, evolutionary algorithms are used in this thesis. In addition, we might introduce a set of constraints, called a language bias, to reduce the search space of rules.

Our research results will be compared and evaluated with state-of-the-art systems in order to identify the weaknesses of the method and to improve it.

1.4 Contributions of the thesis

In this thesis, we address the problem of discovering multi-relational association rules from ontological knowledge bases. Multi-relational association rules are knowledge patterns learned from the ontology and coded in SWRL (Semantic Web Rule Language). This is our answer to the first research question. Unlike ours, some previous approaches (Chapter 2: State-of-the-art) do not consider any background/ontological knowledge and do not exploit any reasoning capabilities. Our discovered rules can be directly added to the ontology, thus enriching its expressive power and augmenting the assertional knowledge that can be derived (Example 1.1). In addition, some discovered rules might be transformed to the representation in DL to complement the ontology (Example 1.2). This is the first contribution of this thesis.

Example 1.1. *Given a knowledge base $\mathcal{K}$ with the following TBox and ABox:*

$$\begin{aligned}\mathcal{T} = \{\ & Male \sqsubseteq Human,\\ & Female \sqsubseteq Human,\\ & \exists.husbandOf.\top \sqsubseteq Male,\\ & \top \sqsubseteq \forall husbandOf.Female\\ & \exists.daughterOf.\top \sqsubseteq Female,\\ & \top \sqsubseteq \forall daughterOf.Human\\ & \exists.motherinlawOf.\top \sqsubseteq Female,\\ & \top \sqsubseteq \forall motherinlawOf.Human\ \}\end{aligned}$$

$$\mathcal{A} = \{\ husbandOf(John,\ Anna),\ husbandOf(Harmen,\ Jasmin)$$

daughterOf(Anna, Flori), daughterOf(Jasmin, Nathalie),
motherinlawOf(Flori, John) }

Suppose we mine the following rule from $\mathcal{K}$:

$$motherinlawOf(x, y) \leftarrow husbandOf(y, z) \wedge daughterOf(z, x)$$

We integrate this rule in $\mathcal{K}$ and retrieve a new assertion, that is motherinlawOf(Nathalie, Harmen)

Example 1.2. *Suppose we discover the following rules from a knowledge base and transform them to the representation in DL:*

1. $Uncle(x) \leftarrow Man(x) \wedge hasBrother(x,y) \wedge hasChild(y,z) \wedge Human(z)$
   becomes
   $Man \sqcap \exists hasBrother.\exists hasChild.Human \sqsubseteq Uncle$ *(concept inclusion)*

2. $hasChild(x,y) \leftarrow hasParent(y,x)$
   becomes
   $hasChild^{-} \equiv hasParent$ *(Inverse role)*

3. $hasSibling(x,y) \leftarrow hasSibling(x,z) \wedge hasSibling(z,y)$
   becomes
   $hasSibling \circ hasSibling \sqsubseteq hasSibling$

In order to answer Research Question 2, in the thesis, we present two algorithms which aim at discovering frequent and accurate hidden patterns in the form of multi-relational association rules. Both algorithms generate rules respecting the language bias (Section 3.5), that is a set of constraints to avoid coming up with redundant or unnecessary rules. In addition, the rules to be discovered by the two algorithms must also satisfy the conditions of basic metrics (Section 3.6). The difference between the two algorithms is as follows:

1. The first algorithm (Chapter 4) discovers all possible rules in the given space of rules (this space is limited by a maximum length of rule and this length is usually short due to system performance problems). The contribution of the first algorithm is, in addition to exploiting reasoning capabilities based on ontological knowledge, to also outperform state-of-the-art ILP (Inductive

Logic Programming) systems in terms of number of discovered rules, using the same samples of the ontologies and the same rule length.

2. The second algorithm (Chapter 5) offers a completely new approach to discover multi-relational association rules, that is to use a genetic algorithm. When using this algorithm, the space of rules might be extended and this means that we can discover rules whose lengths are greater than the lengths of the rules discovered by state-of-the-art systems and the lengths of the rules discovered by the first algorithm without affecting the performance of the system. By considering that a pattern is the genotype of an individual in the population and the corresponding rule is its phenotype, we have defined genetic operators to improve the solutions and fitness functions based on two metrics (the head coverage of the rule and a combination between the head coverage along with confidence of the rule - Section 5.3.3). This contribution has achieved encouraging results because the discovered rules have reached the desired criteria (the number of discovered rules is greater than for the competing systems, the length of rule is arbitrary, discovered rules are consistent with ontological knowledge base).

With the desire to find the best metrics to assess multi-relational association rules and also to answer Research Question 3, we carry out a comparison of popular asymmetric metrics (Chapter 6) adapted from metrics scoring association rules to conform to the Semantic Web. Comparative work is performed by using each metric as a building block for a fitness function for evolutionary inductive programming. The best metrics will be used to score candidate multi-relational association rules in an evolutionary approach to the enrichment of populated knowledge bases in the context of the Semantic Web. The result of this work is an important contribution of the thesis because these metrics have not previously been used in the definition of fitness functions for this purpose.

Calculation speed of metrics is a very important factor while performing experiments on this thesis. We can use SPARQL-DL to query the data needed to assist calculation of metrics. However, the data query speed of SPARQL-DL is pretty slow and does not guarantee the execution in acceptable time, thus we built an algorithm based on the Hash Join algorithm to improve the calculation speed of the metrics. This is also a significant contribution to carry out successful experiments.

1.5 Publications

The following publications have been produced during the thesis implementation process and described the results achieved:

1. Tran Duc Minh, Claudia d'Amato, Binh Thanh Nguyen, Andrea G. B. Tettamanzi. **Comparing Rule Evaluation Metrics for the Evolutionary Discovery of Multi-Relational Association Rules in the Semantic Web**. EuroGP 2018, Parma, Italy, April 4-6, 2018.

2. Tran Duc Minh, Claudia d'Amato, Binh Thanh Nguyen, Andrea G. B. Tettamanzi. **An evolutionary algorithm for discovering multi-relational association rules in the semantic web**. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2017), pages 513-520, Berlin, Germany, July 15-19, 2017.

3. Claudia d'Amato, Andrea G. B. Tettamanzi, Tran Duc Minh. **Evolutionary Discovery of Multi-relational Association Rules from Ontological Knowledge Bases**. Knowledge Engineering and Knowledge Management - 20th International Conference (EKAW 2016), pages 113-128, Bologna, Italy, November 19-23, 2016.

4. Claudia d'Amato, Steffen Staab, Andrea G. B. Tettamanzi, Tran Duc Minh, Fabien L. Gandon. **Ontology enrichment by discovering multi-relational association rules from ontological knowledge bases**. Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC 2016), pages 333-338, Pisa, Italy, April 4-8, 2016.

1.6 Thesis outline

This document is basically divided into three parts. The first part containing the first three chapters presents basic knowledge related to the topic and state of the art. The second part, consisting of chapters 4, 5, 6 and 7, is the detailed content of the thesis. This part details the algorithms, improvements, experimental results, and evaluations in our research. The last part, which is in chapter 8, presents the

conclusions of our research. The contents of the next chapters are summarized as follows:

- **Chapter 2 :** We provide a survey of the state of the art in Semantic Web mining and ontology learning.
- **Chapter 3 :** This chapter is devoted to the basic knowledge related to this thesis and is composed of two parts. The first part presents some fundamental knowledge of the Semantic Web, the second part offers notations and formal definitions that are used to support algorithms and to evaluate generated rules.
- **Chapter 4 :** We propose an algorithm whose aim is to discover hidden knowledge patterns in the form of multi-relational association rules by implementing the level-wise generate-and-test approach.
- **Chapter 5 :** We propose a genetic algorithm that is an improvement of the algorithm in Chapter 4 and on the state-of-the-art systems for the purpose of the work.
- **Chapter 6 :** We adapt popular asymmetric metrics proposed for scoring association rules to metrics that match the Semantic Web. And in this chapter, we compare these metrics by considering them as building blocks for the fitness function of the genetic algorithm mentioned in Chapter 5 in order to select suitable metrics to score multi-relational association rules.
- **Chapter 7 :** In order to improve the calculation of metrics, in this chapter, we propose an algorithm based on the Hash Join algorithm but adjusted to fit the generated SWRL rules. This algorithm replaces SPARQL-DL query encountering problems in performance.
- **Chapter 8 :** This chapter offers conclusions based on the contents of the previous chapters and it also provides perspectives and future work to improve our research results.

Chapter 2

State of the Art Review

Contents

2.1 Semantic Web Mining 11
2.2 Expert Rule Mining 12
2.3 Ontology learning 12
2.3.1 Concept learning 12
2.3.2 Pattern mining 13
2.3.3 Evolutionary algorithm for learning 14
2.4 Pattern evaluation 14

2.1 Semantic Web Mining

The integration of the two scientific research areas of Semantic Web and Web Mining is known as Semantic Web Mining. An early state-of-the-art survey on Semantic Web Mining [SHB06], dating back to 2006, analyzes the convergence of trends from these two research areas, it then sketches ways of how a closer integration could be profitable. Recently, the huge increase in the amount of Semantic Web data became a perfect target for many researchers to apply Data Mining techniques on it. Another state-of-the-art survey on Semantic Web Mining in 2013 [QKQ13] gives a detailed account of the advances in this new research area. It shows the positive effects of Semantic Web Mining, the obstacles faced by researchers and proposes a number of approaches to deal with the very complex and heterogeneous information and knowledge which are produced by the technologies of the Semantic Web. A recent survey (in 2016) in addition to providing various classifications of web mining along with its subtasks, gives a perspective to the research community about the potential of applying techniques to extract meaningful patterns [T.R16].

2.2 Expert Rule Mining

An expert approach is proposed to discover causal relations in RDF-based medical data [NB12]. This method requires a domain expert defining contexts and objectives of mining and extraction process.

2.3 Ontology learning

Building ontologies manually is extremely labor-intensive and time-consuming. An approach is called Ontology Learning that might go from the simple extraction to the induction of description logic. Some of the researches we can refer to are:

- Learning ontologies from RDF annotations of Web resources [DFZD01]. Initially, this method extracts some of resource descriptions from the whole RDF graph gathering all the annotations. Then it builds ontology by gradually increasing the size of the resource descriptions.
- Clustering ontology-based metadata in the Semantic Web [MZ02]. This approach defines a set of similarity measures that allow to compute similarities between ontology-based metadata along different dimensions. These measures are then applied within a hierarchical clustering algorithm.
- Some techniques in the ontology learning cycle have been implemented in KAON Text-To-Onto [MS]. This ontology learning framework proceeds through ontology import, extraction, pruning, and refinement to give the ontology engineer a wealth of coordinated tools for ontology modeling.
- Learning Meta-Descriptions by using clustering to identify classes of people and Inductive Logic Programming (ILP) to learn descriptions of these groups in the FOAF Network [GEP04].

2.3.1 Concept learning

The W3C Web Ontology Language (OWL) is designed to represent rich and complex knowledge about things, groups of things, and relations between things. The following approaches have been developed to automatically learn concepts and produce their descriptions in OWL:

- DL-Learner [Leh09], a framework for learning concepts in description logics and OWL, includes several Machine Learning algorithms, support for different OWL formats, reasoner interfaces, and learning problems in OWL.

- An approach for mining and analyzing large knowledge bases based on DL-Learner [HLA08]. This approach obtains complex class descriptions from objects in large knowledge bases available as SPARQL endpoints or Linked Data by using Machine Learning techniques.

- A FOIL-like algorithm is proposed that can be applied to general DL languages [FdE08b] and is implemented in the DL-FOIL system. The main components of this system are represented by a set of refinement operators and by a different gain function which takes into account the open world assumption.

2.3.2 Pattern mining

Pattern mining consists of using data mining algorithms to learn (to discover) useful and structured patterns. The following researches proposed solutions for mining hidden knowledge patterns in the Semantic Web:

- A solution focuses on the relation of the semantics of the representation formalism to the task of frequent pattern discovery and exploits the semantics of the combined knowledge base to perform semantic tests [JLL10a]. This method is used to discover DATALOG clauses and patterns having the form of conjunctive queries over the combined knowledge base. It is grounded on a notion of key, standing for the basic attribute to be used for counting elements for building the frequent patterns.

- A statistical approach to the induction of expressive schemas from large RDF datasets is described in [VN11]. The kind of knowledge that is explored in this approach is association rules. The mined association rules can be translated into OWL 2 EL axioms in a relatively straightforward way. However, this approach does not exploit any reasoning capabilities.

- Another approach for inducing new assertional knowledge from RDF datasets is presented in [GTHS13]. This approach also mines association rules and does not apply any reasoning capabilities. However, it develops a rule mining

model that is explicitly tailored to support the open world assumption (OWA) scenario.

2.3.3 Evolutionary algorithm for learning

Learning can be viewed as a search problem in the space of all possible patterns. This problem is usually NP-hard, however, evolutionary algorithm approaches keep a good balance between exploration and exploitation of the solution spaces. Several previous researches have applied evolutionary algorithms to learn concepts:

- An approach applied an evolutionary algorithm to learn concepts for description logics by combining refinement operators and Genetic Programming [Leh06].
- An evolutionary algorithm for concept learning in First Order Logic (FOL) was also introduced [DM02], which evolves a population of Horn clauses by repeated selection, mutation and optimization of more fit clauses. A new point in this technique is that the use of stochastic search biases for reducing the complexity of the search process and of the clause fitness evaluation.
- A state-of-the-art survey provides an overview of evolutionary techniques for ILP to learn concepts in FOL [Div06]. Six systems (Regal, G-Net, Dogma, Sia01, ECL and GLPS) are described and compared by means of the following aspects: search strategy, representation, evaluation, search operators and biases adopted for limiting the search space.

2.4 Pattern evaluation

Evaluation of the quality of the patterns generated by the mining (learning) techniques is an important step to identify the most interesting patterns. Therefore, we need to be aware of the importance of choosing appropriate measures for our study. There are quite a number of objective measures used to assess the quality of the patterns and some researches performing to compare these measures to aim at selecting the right ones as follows:

- A comparative study of several key properties in order to select the right measure for a given application [TKS04]. This research has compared proper-

ties based on twenty-one measures for association patterns and it shows that each measure might be appropriate for some applications but not for other applications.

– Another comparison of metrics for mining rules from data (the metrics including confidence, support, gain, chi-squared value, gini-index, entropy gain, laplace, lift, and conviction) shows that the best rule according to any of these metrics must reside along a support/confidence border [BA99].

– A research of the predictive ability of some association rule measures [AJ07] performed an evaluation on seventeen datasets. The research concluded that conviction is on average the best predictive measure.

All the above researches compare the metrics where the CWA (Closed World Assumption) is used along with a platform based on statistical inference. There are some other metrics proposed as being appropriate for a scenario with OWA (Open World Assumption) that the Semantic Web works under.

1. AMIE [GTHS13] has proposed a metric called PCA-Confidence (where PCA stands for Partial Completeness Assumption). This metric assumes that, in all role assertions, if we know one object for a given subject and predicate (role name), then we know all objects for that subject and predicate. This assumption allow us to generate counter-examples to increase the accuracy of the metric, specifically here it is the confidence metric to be focused on.

2. A metric based on possibility theory is proposed by [TFZG14], where a candidate axiom holds two values: a degree of possibility and a degree of necessity. This metric is suitable to represent incomplete knowledge bases and to be used for automatic axiom induction from ontology. In addition, it also provides a solid foundation for learning ontology.

Chapter 3

Background

Contents

3.1 Introduction 17
3.2 Inductive Logic Programming 18
3.3 Description Logics 19
3.3.1 Knowledge Base 19
3.3.2 Semantics 22
3.3.3 Web Ontology Language OWL 24
3.4 Semantic Web Rule Language (SWRL) 28
3.5 Language Bias 29
3.6 Basic metrics for rules evaluation 32
3.7 Extensive metrics for rules evaluation 35
3.8 Evaluation of Rule Precision 37

3.1 Introduction

In this chapter, we will cover some basic concepts relating to the content of the next chapters.

Initially, Inductive Logic Programming (ILP) is introduced in Section 3.2. Next, in Section 3.3, we present the basic notations and terminology related to the semantic technologies for the Web of data, which is the basis for our research problem that is to discover hidden knowledge patterns in the form of multi-relational association rules coded in Semantic Web Rule Language (SWRL). Therefore, in Section 3.4, we focus on introducing SWRL.

In Sections 3.5 and 3.6, we mention fundamental issues of how to discover hidden knowledge patterns effectively. In Section 3.5, we propose a language bias used to

reduce the search space of rules and generate non-redundant rules; In Section 3.6, we present metrics used to evaluate the quality of generated rules. In addition, we come up with an extensive list of metrics borrowed from scoring association rules to apply to the assessment of generated rules in our work (Section 3.7). In order to measure and evaluate the ability of a rule to perform correct predictions, we carry out experiments together with metrics shown inside Section 3.8.

As described in Section 3.3.1, we refer to an ontological DL knowledge base containing a set of axioms, which are of two kinds: terminological (TBox) and assertional (ABox). In the following the general definition of a relational association rule for an ontological knowledge base is given. Hence, the problem we want to address is defined.

Definition 1 (Relational Association Rule)**.** *Given a populated ontological KB* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$*, a* relational association rule *r for* $\mathcal{K}$ *is a Horn-like clause of the form: body* $\rightarrow$ *head, where: (a) body is a generalization of a set of assertions in* $\mathcal{K}$ *co-occurring together; (b) head is a consequent that is induced from* $\mathcal{K}$ *and body*

Definition 2 (Problem Definition)**.**

Given:

- *a populated ontological knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{A})$*;*
- *a minimum "frequency threshold",* θ_f*;*
- *minimum "metrics thresholds";*

Discover: *all frequent and fit hidden patterns with respect to* θ_f *and "metrics thresholds", in the form of relational association rules, that may induce new assertions for* $\mathcal{K}$*.*

Intuitively, a *frequent hidden pattern* is a generalization of a set of concept/-role assertions co-occurring reasonably often (with respect to a fixed frequency threshold) together, showing an underlying form of correlation that is exploited for obtaining new assertions.

3.2 Inductive Logic Programming

Inductive Logic Programming (ILP) is a research field essentially combining Machine Learning and Logic Programming, which investigates the inductive construc-

tion of logic programs from examples and background knowledge ([Mug91], [Dž96], [MR94], [MDRP+12]). Learning from datasets to build logic programs is the main task of ILP because the logic program provides a good representation of the generalization of a issue suggested to make predictions. In addition, the logic programs are also more expressive than alternative representations such as network and graph-based representations.

Learning logic programs is considered as a search problem in the space of all possible solutions. This problem is often NP-hard, thus we need to figure out the right solution to decrease the search space and in combination with evaluating the complexity of the solutions.

3.3 Description Logics

Description logics (DLs) [BCM+03] are a family of knowledge representation languages that are widely used in ontological modelling.

We use description logics to represent knowledge about a domain of interest; description logics are based on three disjoint sets of primal elements:

1. *Concept names* contains names that refer to types, categories or classes of entities. For instance: Country, University, Animal, Doctor, ...

2. *Role names* contains names that denote binary relationships which might hold between individuals of domain. For instance: sonOf, fatherOf, hasChild, livedIn, ...

3. *Individual names* contains names that denote singular entities in the domain of interest. For instance: the sun, JOHN, ...

3.3.1 Knowledge Base

A knowledge representation system based on description logics provides facilities to set up knowledge bases, to reason about their content, and to manipulate them. A knowledge base DL does not fully describe a specific situation, instead it includes a set of statements, called *axioms*, each of which must be true in the situation described. Often these axioms only capture a partial knowledge of the situation that the ontology is describing and at the same time they are consistent with that

ontology. A knowledge base separates *axioms* into three components: terminological (TBox) axioms, assertional (ABox) axioms and relational (RBox) axioms. In this thesis, when we mention TBox, this means that we refer to both TBox and RBox.

– The TBox is considered as intensional knowledge in the form of sentences relating concepts (terms) to other concepts, which introduces the terminology, i.e., the vocabulary of an application domain.

  Let C and D be concepts. Then:

  * *Concept inclusion* has the form $C \sqsubseteq D$ that states that every C is a D. This type of statement is also called *subsumption* and $C \sqsubseteq D$ is often read "C is subsumed by D" or "D subsumes C".
  * *Concept equivalence* has the form $C \equiv D$ that asserts that two concepts C and D have the same instances. This kind of statement is regarded as an abbreviation for the two concept inclusions $C \sqsubseteq D$ and $D \sqsubseteq C$. $C \equiv D$ is often read "C and D are equivalent".

  Example 3.1 (TBox)**.** *TBox defines concepts of the application domain, their properties and their relations, actually inclusions, to each other:*

  $Male \sqsubseteq Human$
  $Female \sqsubseteq Human$
  $Male \sqsubseteq \neg Female$
  $Mother \sqsubseteq Human \sqcap Female$
  $Parent \sqsubseteq Human \sqcap \exists fatherOf.Human$
  $fatherOf \sqsubseteq parentOf$

– The ABox is regarded as extensional knowledge, which contains assertions about named individuals in terms of this vocabulary.

  Example 3.2 (ABox)**.** *ABox makes assertions about the individuals in the application domain:*

  Female(ANNA)
  motherOf(ANNA, JOHN)

– The RBox refer to properties of roles. In addition, it also refers to characteristics of roles such as transitive, symmetric, asymmetric, reflexive, irreflexive, functional, inverse functional.

Let r and s be roles. Then

* *Role inclusion* has the form $r \sqsubseteq s$ that states that every pair of individuals related by r is also related by s. This type of statement is also called *subrole* and $r \sqsubseteq s$ is often read "r is a subrole of s".

Example 3.3 (RBox)**.** *role inclusion and disjoint role:*

fatherOf $\sqsubseteq$ *parentOf*
Disjoint(spouseOf, siblingOf)

Starting with *atomic concepts* and *atomic roles*, which are simply described by concept names and role names respectively, *complex concepts* (called concepts) are inductively defined by expressions built using suitable constructs (column 2 of Table 3.1) as follows:

– Every concept name is a concept.

– $\top$, which is the common super type of all defined concepts in KB and captures all individuals in the domain, is a concept (called *top concept* or *Thing*).

– $\bot$, which is the empty set or nothing, is a concept (called *bottom concept*)

– If C and D are concepts, then $\neg C$ (complement), $C \sqcap D$ (intersection), $C \sqcup D$ (union) are also concepts.

– If r is a role and C is a concept then so are $\exists R.C$ (existential restriction), $\forall R.C$ (universal restriction).

– If r is a atomic role, n is a non-negative integer and C is a concept, then $\exists R.Self$ (local reflexivity), $\geqslant n\ R.C$ (at-least restriction) and $\leqslant n\ R.C$ (at-most restriction) are also concepts.

– Every finite set $\{a_1, ..., a_n\} \subseteq N_I$ (the set of individual names) is a concept; concepts of this type are called *nominal concepts*.

Example 3.4 (Complex concepts)**.** *The following expressions are complex concepts (also called concepts):*

- $Human \sqcap Female$
- $Human \sqcap \exists fatherOf.Human$

DLs to be used in this thesis are based on description logic $\mathcal{SROIQ}$ [HKS06] which is one of the most expressive DLs commonly considered today. $\mathcal{SROIQ}$ roughly corresponds to the set of constructors available in OWL 2 (Section 3.3.3). $\mathcal{SROIQ}$ is constituted by fragments as follows:

- $\mathcal{SR}$ denotes description logic $\mathcal{ALC}$ extended with all kinds of RBox axioms as well as self concepts.
  - $\mathcal{ALC}$ (Attribute Language with general Complement) only allows atomic concepts, $\top$, $\bot$, $\neg$, $\sqcap$, $\sqcup$, $\exists$, $\forall$ as its concept constructors, but do not allow RBox axioms.
- $\mathcal{O}$ indicates that nominal concepts are assisted.
- $\mathcal{I}$ indicates that role inverses are supported.
- $\mathcal{Q}$ indicates that qualified number restrictions are supported.

3.3.2 Semantics

The semantics of description logics is given by a model-theoretic way. The semantics specifies what the logical consequences of an ontology are. The purpose of the semantics is to give a consequence relation, which tells us whether an axiom is a logical consequence of a KB. Therefore, one central notion is that of an interpretation, normally denoted with a symbol $\mathcal{I}$, which is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consisting of:

1. a nonempty set $\Delta^{\mathcal{I}}$, called the domain or universe of discourse, which is regarded as the whole of individuals or things existing in the domain that $\mathcal{I}$ represents.
2. a function $\cdot^{\mathcal{I}}$, called interpretation function, which maps the vocabulary elements to $\Delta^{\mathcal{I}}$. Specifically, it provides:

Table 3.1: Syntax and semantics of $\mathcal{SROIQ}$ constructors

	Syntax	Semantics
individual name	a	$a^{\mathcal{I}}$
atomic concept	C	$C^{\mathcal{I}}$
intersection	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
union	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
complement	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
top concept	$\top$	$\Delta^{\mathcal{I}}$
bottom concept	$\bot$	$\emptyset$
existential restriction	$\exists R.C$	$\{x \mid$ some $R^{\mathcal{I}}$ -successor of x is in $C^{\mathcal{I}}\}$
universal restriction	$\forall R.C$	$\{x \mid$ all $R^{\mathcal{I}}$ -successors of x are in $C^{\mathcal{I}}\}$
at-least restriction	$\geqslant n\ R.C$	$\{x \mid$ at least n $R^{\mathcal{I}}$ -successors of x are in $C^{\mathcal{I}}\}$
at-most restriction	$\leqslant n\ R.C$	$\{x \mid$ at most n $R^{\mathcal{I}}$ -successors of x are in $C^{\mathcal{I}}\}$
local reflexivity	$\exists R.Self$	$\{x \mid (x,x) \in R^{\mathcal{I}}\}$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
atomic role	R	$R^{\mathcal{I}}$
inverse role	R^{-}	$\{(x,y) \mid (y,x) \in R^{\mathcal{I}}\}$
universal role	U	$\Delta^{\mathcal{I}}$ x $\Delta^{\mathcal{I}}$
where a, b are individual names, A is a concept name, C, D are concepts, R is a role "$R^{\mathcal{I}}$ -successors of x" means any individual y such that (x, y) $\in R^{\mathcal{I}}$		

- every individual name a to an element $a^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$.
- every concept name C to a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$.
- every role name R to a subset $R^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ x $\Delta^{\mathcal{I}}$.

The interpretation of complex concepts and roles follows from the interpretation of the basic expressions. Table 3.2 shows the way to achieve the semantics of each compound expression from the semantics of its components.

In addition, the consequence relation is denoted by $\models$ and defined as follows: An axiom $\mathcal{A}$ is a consequence of (satisfied by) a knowledge base $\mathcal{KB}$ (written $\mathcal{KB} \models \mathcal{A}$) if every model of the $\mathcal{KB}$ is also a model of $\mathcal{A}$.

* **Satisfaction**: Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be an interpretation.

- $\mathcal{I}$ satisfies the statement $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (written $\mathcal{I} \models C \sqsubseteq D$).
- $\mathcal{I}$ satisfies the statement $C \equiv D$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$ (written $\mathcal{I} \models C \equiv D$).
- $\mathcal{I}$ satisfies $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ (written $\mathcal{I} \models C(a)$).
- $\mathcal{I}$ satisfies $R(a,b)$ if $(a^{\mathcal{I}},b^{\mathcal{I}}) \in R^{\mathcal{I}}$ (written $\mathcal{I} \models R(a,b)$).

Table 3.2: Syntax and semantics of $\mathcal{SROIQ}$ axioms

	Syntax	Semantics
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$R(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
individual equality	$a \approx b$	$a^{\mathcal{I}} = b^{\mathcal{I}}$
individual inequality	$a \not\approx b$	$a^{\mathcal{I}} \neq b^{\mathcal{I}}$
concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
concept equivalence	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
role inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
role equivalence	$R \equiv S$	$R^{\mathcal{I}} = S^{\mathcal{I}}$
complex role inclusion	$R_1 \circ R_2 \sqsubseteq S$	$R_1^{\mathcal{I}} \circ R_2^{\mathcal{I}} \subseteq S$
role disjointness	$Disjoint(R, S)$	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$

* **Model**:
  - An interpretation $\mathcal{I}$ is a model for a TBox $\mathcal{T}$ if $\mathcal{I}$ satisfies all the statements in $\mathcal{T}$.
  - An interpretation $\mathcal{I}$ is a model for a ABox $\mathcal{A}$ if $\mathcal{I}$ satisfies every assertion of $\mathcal{A}$.
* **Satisfiability**:
  - A TBox $\mathcal{T}$ is satisfiable if it has a model.
  - An ABox $\mathcal{A}$ is satisfiable if it has a model.

3.3.3 Web Ontology Language OWL

The Web Ontology Language OWL is a component of the Semantic Web activity, which is a W3C standard, builds on RDF and RDFS to let us define ontologies. This aims to make Web resources more readily accessible to automated processes.

In addition, OWL makes an open world assumption (OWA). The OWA means that what cannot be inferred from the knowledge base might be true as well as false, contrary to what holds for closed world assumption (CWA), where everything that is not stated in the CWA is false.

In order to write an ontology that can be interpreted unambiguously and used by software agents we require a syntax and formal semantics for OWL. OWL is a

vocabulary extension [RDF Semantics] of RDF. The OWL semantics are defined in OWL Web Ontology Language Semantics and Abstract Syntax.

OWL has three species: OWL-Lite, OWL DL, and OWL Full.

- OWL-Lite is a variant of the description logic $\mathcal{SHIF}(\mathcal{D})$. In which, $\mathcal{F}$ at the end of a DL name allows for role functionality statements which can be expressed as $\top \sqsubseteq \leqslant 1.\top$; $(\mathcal{D})$ enables use of datatype properties, data values or data types.

- OWL DL is a variant of the description logic $\mathcal{SHOIN}(\mathcal{D})$. In which, $\mathcal{N}$ at the end of a DL name enables assist for unqualified number restrictions, i.e., concepts of the form $\geqslant nr.\top$ and $\leqslant nr.\top$.

- OWL Full is based on a different semantics from OWL Lite or OWL DL, and was designed to preserve some compatibility with RDF Schema. OWL Full allows an ontology to augment the meaning of the pre-defined vocabulary. OWL Full is undecidable, so no reasoning software is able to perform complete reasoning for it.

For instance, column 1 and 2 of Figure 3.1 and Figure 3.2 for a mapping between the OWL DL abstract syntax and the syntax of the description logic $\mathcal{SHOIN}(\mathcal{D})$.

A new version of OWL is OWL 2 [W3Cb] which has a very similar overall structure to OWL. OWL 2 is backwards compatibility with OWL, all OWL ontologies remain valid OWL 2 ontologies, with identical inferences in all practical cases.

OWL 2 adds a couple of new features with respect to OWL as follows (some features are syntactic sugar) :

- keys

- property chains

- richer datatypes, data ranges

- qualified cardinality restrictions

- asymmetric, reflexive, and disjoint properties

- enhanced annotation capabilities

Abstract Syntax	DL Syntax	Semantics
Descriptions (C)		
A (URI Reference)	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
`owl:Thing`	$\top$	$\texttt{owl:Thing}^{\mathcal{I}} = \Delta^{\mathcal{I}}$
`owl:Nothing`	$\bot$	$\texttt{owl:Nothing}^{\mathcal{I}} = \emptyset$
`intersectionOf`$(C_1\ C_2 \ldots)$	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
`unionOf`$(C_1\ C_2\ \ldots)$	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
`complementOf`(C)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
`oneOf`$(o_1 \ldots)$	$\{o_1, \ldots\}$	$\{o_1^{\mathcal{I}}, \ldots\}$
`restriction`$(R$ `someValuesFrom`$(C))$	$\exists R.C$	$\{x \mid \exists y\ \ (x,y) \in R^{\mathcal{I}} \cup y \in C^{\mathcal{I}}\}$
`restriction`$(R$ `allValuesFrom`$(C))$	$\forall R.C$	$\{x \mid \forall y\ \ (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
`restriction`$(R$ `hasValue`$(o))$	$R : o$	$\{x \mid (x, o^{\mathcal{I}}) \in R^{\mathcal{I}}\}$
`restriction`$(R$ `minCardinality`$(n))$	$\geqslant nR$	$\{a \in \Delta^{\mathcal{I}} \mid\ \mid\{b \mid (a,b) \in R^{\mathcal{I}}\}\mid \geq n\}$
`restriction`$(R$ `maxCardinality`$(n))$	$\leqslant nR$	$\{a \in \Delta^{\mathcal{I}} \mid\ \mid\{b \mid (a,b) \in R^{\mathcal{I}}\}\mid \leq n\}$
`restriction`$(U$ `someValuesFrom`$(D))$	$\exists U.D$	$\{x \mid \exists y\ \ (x,y) \in U^{\mathcal{I}} \cup y \in D^{\mathcal{D}}\}$
`restriction`$(U$ `allValuesFrom`$(D))$	$\forall U.D$	$\{x \mid \forall y\ \ (x,y) \in U^{\mathcal{I}} \rightarrow y \in D^{\mathcal{D}}\}$
`restriction`$(U$ `hasValue`$(v))$	$U : v$	$\{x \mid (x, v^{\mathcal{I}}) \in U^{\mathcal{I}}\}$
`restriction`$(U$ `minCardinality`$(n))$	$\geqslant nU$	$\{a \in \Delta^{\mathcal{I}} \mid\ \mid\{b \mid (a,b) \in U^{\mathcal{I}}\}\mid \geq n\}$
`restriction`$(U$ `maxCardinality`$(n))$	$\leqslant nU$	$\{a \in \Delta^{\mathcal{I}} \mid\ \mid\{b \mid (a,b) \in U^{\mathcal{I}}\}\mid \leq n\}$
Data Ranges (D)		
D (URI reference)	D	$D^{\mathcal{D}} \subseteq \Delta_{\mathcal{D}}^{\mathcal{I}}$
`oneOf`$(v_1 \ldots,)$	$\{v_1 \ldots,\}$	$\{v_1^{\mathcal{I}} \ldots,\}$
Object Properties (R)		
R (URI reference)	R	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	R^-	$(R^{\mathcal{I}})^-$
Datatype Properties (U)		
U (URI reference)	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$
Individuals (o)		
o (URI reference)	o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Data Values (v)		
v (RDF literal)	v	$v^{\mathcal{D}}$

Figure 3.1: OWL DL descriptions, data ranges, properties, individuals and data values syntax and semantics [Obi]

Abstract Syntax	DL Syntax	Semantics
Classes		
Class(A partial $C_1 \dots C_n$)	$A \sqsubseteq C_1 \sqcap \dots \sqcap C_n$	$A^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \cap \dots \cap C_n^{\mathcal{I}}$
Class(A complete $C_1 \dots C_n$)	$A \equiv C_1 \sqcap \dots \sqcap C_n$	$A^{\mathcal{I}} = C_1^{\mathcal{I}} \cap \dots \cap C_n^{\mathcal{I}}$
EnuneratedClass(A $o_1 \dots o_n$)	$A \equiv \{o_1, \dots, o_n\}$	$A^{\mathcal{I}} = \{o_1^{\mathcal{I}}, \dots, o_n^{\mathcal{I}}\}$
SubClassOf(C_1 C_2)	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
EquivalentClasses($C_1 \dots C_n$)	$C_1 \equiv \dots \equiv C_n$	$C_1^{\mathcal{I}} = \dots = C_n^{\mathcal{I}}$
DisjointClasses($C_1 \dots C_n$)	$C_i \sqcap C_j = \bot, i \neq j$	$C_i^{\mathcal{I}} \cap C_j^{\mathcal{I}} = \emptyset, i \neq j$
Datatype(D)		$D^{\mathcal{C}} \Delta_{\mathcal{D}}^{\mathcal{I}}$
Datatype Properties		
DatatypeProperty(		
U super(U_1) ... super(U_n)	$U \sqsubseteq U_i$	$U^{\mathcal{I}} \subseteq U_i^{\mathcal{I}}$
domain(C_1) ... domain(C_m)	$\geqslant 1\, U \sqsubseteq C_i$	$U^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$
range(D_1) ... range(D_l)	$\top \sqsubseteq \forall U.D_i$	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times D_i^{\mathcal{I}}$
[Functional])	$\top \sqsubseteq \leqslant 1 U$	U_i is functional
SubPropertyOf(U_1 U_2)	$U_1 \sqsubseteq U_2$	$U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$
EquivalentProperties($U_1 \dots U_n$)	$U_1 \equiv \dots \equiv U_n$	$U_1^{\mathcal{I}} = \dots = U_n^{\mathcal{I}}$
Object Properties		
ObjectProperty(		
R super(R_1) ... super(R_n)	$R \sqsubseteq R_i$	$R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}$
domain(C_1) ... domain(C_m)	$\geqslant 1\, R \sqsubseteq C_i$	$R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$
range(C_1) ... range(C_l)	$\top \sqsubseteq \forall R.C_i$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_i^{\mathcal{I}}$
[inverseOf(R_0)]	$R \equiv (R_0^-)$	$R^{\mathcal{I}} = (R_0^{\mathcal{I}})^-$
[Symmetric]	$R \equiv (R^-)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^-$
[Functional]	$\top \sqsubseteq \leqslant 1 R$	$R^{\mathcal{I}}$ is functional
[InverseFunctional]	$\top \sqsubseteq \leqslant 1 R^-$	$(R^{\mathcal{I}})^-$ is functional
[Transitive])	$Tr(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$
SubPropertyOf(R_1 R_2)	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
EquivalentProperties($R_1 \dots R_n$)	$R_1 \equiv \dots \equiv R_n$	$R_1^{\mathcal{I}} = \dots = R_n^{\mathcal{I}}$
Annotation		
AnnotationProperty(S)		
Individuals		
Individual(		
o type(C_1) ... type(C_n)	$o \in C_i$	$o^{\mathcal{I}} \in C_i^{\mathcal{I}}$
value(R_1 o_1) ... value(R_n o_n)	$\{o, o_i\} \in R_i$	$\{o^{\mathcal{I}}, o_i^{\mathcal{I}}\} \in R_i^{\mathcal{I}}$
value(U_1 v_1) ... value(U_n v_n))	$\{o, v_i\} \in U_i$	$\{o^{\mathcal{I}}, v_i^{\mathcal{I}}\} \in U_i^{\mathcal{I}}$
SameIndividual($o_1 \dots o_n$)	$o_1 = \dots = o_n$	$o_1^{\mathcal{I}} = \dots = o_n^{\mathcal{I}}$
DifferentIndividual($o_1 \dots o_n$)	$o_i \neq o_j, i \neq j$	$o_i^{\mathcal{I}} \neq o_j^{\mathcal{I}}, i \neq j$

Figure 3.2: OWL DL axioms and facts [Obi]

OWL 2 also defines a new syntax (called Manchester syntax [W3Ca]) and three new profiles OWL 2 EL, OWL 2 QL and OWL 2 RL [W3Cc]. These profiles have advantages in particular application scenarios:

- OWL 2 EL enables polynomial time algorithms for all the standard reasoning tasks; it is particularly suitable for applications where very large ontologies are needed, and where expressive power can be traded for performance guarantees.
- OWL 2 QL is based on description logics similar to DL-Lite. It enables conjunctive queries to be answered in LogSpace (more precisely, AC^0) using standard relational database technology; it is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to access the data directly via relational queries (e.g., SQL).
- OWL 2 RL enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies operating directly on RDF triples; it is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to operate directly on data in the form of RDF triples.

3.4 Semantic Web Rule Language (SWRL)

SWRL is an acronym for Semantic Web Rule Language, which is a standard language based on OWL-DL and on the Rule Markup Language (RuleML) which provides both OWL-DL expressivity and rules from RuleML [HPSB+04]. SWRL represents Horn-like rules expressed in terms of OWL concepts to reason about OWL individuals. Rules in SWRL are implication rules and they can be used to infer new knowledge from the existing OWL knowledge bases, which is an important feature that we are interested in in this work.

A SWRL rule has the following form [Mun10]:

$$\text{consequent} \leftarrow \text{antecedent}$$

This syntax shows that the consequent must be true when the antecedent is satisfied. The antecedent is referred to as the rule body and the consequent is referred

to as the head in the Definition 1. The head and body consist of a conjunction of one or more atoms.

Definition 3 (SWRL Rule)**.** *Given a knowledge base $\mathcal{K}$, a SWRL rule is an implication between an antecedent (body) and a consequent (head) of the form:* $H_1 \wedge \cdots \wedge H_m \leftarrow B_1 \wedge \cdots \wedge B_n$, *where* $B_1 \wedge \cdots \wedge B_n$ *is the rule body and* $H_1 \wedge \cdots \wedge H_m$ *is the rule head. Each* $B_1, \ldots, B_n, H_1, \ldots H_m$ *is called an* atom.

An atom *is a unary or binary predicate of the form* $C(s)$, $R(s_1, s_2)$, $\mathsf{sameAs}(s_1, s_2)$ *or* $\mathsf{differentFrom}(s_1, s_2)$, *where the predicate symbol* C *is a concept name in* $\mathcal{K}$, R *is a role name in* $\mathcal{K}$, s, s_1, s_2 *are* terms. *A* term *is either a variable (denoted by* x, y, z*) or a constant (denoted by* a, b, c*) standing for an individual name or data value. Variables are treated as universally quantified, with their scope limited to a given rule.*

The SWRL rules can be generally called *multi-relational* rules since multiple binary predicates $R(s_1, s_2)$ with different role names of $\mathcal{K}$ could appear in a rule. A rule having more than one atom in the head can be equivalently transformed, due to the *safety condition* (see Definition 4), into multiple-relational rules, each one having the same body and a single atom in the head. In our work, we will only consider SWRL rules with one atom in the head.

Example 3.5 (SWRL rule)**.** *Given a SWRL rule of the form* $\mathsf{sonOf}(x, y) \leftarrow \mathsf{fatherOf}(y, x) \wedge \mathsf{Male}(x)$ *where* $\mathsf{sonOf}(x, y)$ *is the rule head;* $\mathsf{fatherOf}(y, x) \wedge \mathsf{Male}(x)$ *is the rule body;* fatherOf*,* Male*,* sonOf *are atoms and* x, y *are variables.*

3.5 Language Bias

We know that the search space of the rules formed in SWRL is huge. Thus, in order to reduce the search space, we use a set of constraints giving a tight specification of the patterns worth considering. This set is called a language bias. In order to manage language bias, we are interested in the following aspects:

– Just focus on the rules given by a set of atomic concept names (atomic unary predicates), a set of atomic role names (atomic binary predicates) and a set of individuals (functional constants).

– Only consider the connected rules [GTHS13] to be defined in Definition 5 and non-redundant rules [JLL10b] to be defined in Definition 7. Additionally, the considered rules must satisfy the safety condition [HPS04] to be defined in Definition 4.

– In order to guarantee decidability which means that a conclusion can always be reached, only DL-safe rules are managed [MSS05], that is rules interpreted under the DL-safety condition consisting in binding all variables in a rule only to explicitly named individuals in knowledge base. When added to an ontology, DL-safe rules are decidable and generate sound results but not necessarily complete.

Given an atom A, let $T(A)$ denote the set of all the terms occurring in A and let $V(A) \subseteq T(A)$ denote the set of all the variables occurring in A.

Example 3.6. *Suppose that C is an atomic concept, R is an atomic role. We have: $V(C(x)) = \{x\}$ and $V(R(x, y)) = \{x, y\}$*

Definition 4 (Safety Condition)**.** *Given a knowledge base $\mathcal{K}$ and a rule $r = H \leftarrow B_1 \wedge B_2 \wedge \cdots \wedge B_n$, r satisfies the* safety condition *if all variables appearing in the rule head also appear in the rule body; formally if: $V(H) \subseteq \bigcup_{i=1}^{n} V(B_i)$,*

The constraint to safety condition of the rule avoids mining meaningless rules.

Example 3.7. *Rule spouseOf(x, y) ← fatherOf(x, z) ∧ Female(z) does not satisfy the* safety condition*, since the variable y appears in the head but not appear in the body.*

Definition 5 (Connected Rule)**.** *Given a knowledge base $\mathcal{K}$ and a rule $r = H \leftarrow B_1 \wedge B_2 \wedge \cdots \wedge B_n$, r is* connected *if and only if every atom in r is transitively connected to every other atom in r.*

Two atoms B_i and B_j in r, with $i \neq j$, are connected *if they share at least a variable or a constant i.e. if $T(B_i) \cap T(B_j) \neq \emptyset$.*

Two atoms B_1 and B_k in r are transitively connected *if in r there exist atoms $B_2, \ldots, B_{k-1}$, with $k \leq n$, such that, for all $i, j \in \{1, \ldots, k\}$ with $i \neq j$, $T(B_i) \cap T(B_j) \neq \emptyset$, in which n is the total number of atoms in r.*

The constraint to connected rules avoids mining rules with completely unrelated atoms.

Example 3.8. *Rule liveIn(x, y) ← workIn(w, v) is not a* connected rule*, since two atoms in the rule are completely unrelated.*

Definition 6 (Closed Rule)**.** *Given a knowledge base $\mathcal{K}$ and a rule $r = H \leftarrow B_1 \wedge B_2 \wedge \dots B_n$, r is* closed *if and only if every variable in r is closed.*

Each variable $v_j \in \bigcup_{i=0}^{n} V(B_i), j \in \{0, \dots, k\}$, with $k \leq n$, $B_0 = H$, is closed if it appears at least twice in r, in which n is the total number of atoms in r.

The constraint to closed rules avoids mining rules that predict merely the existence of a fact.

Example 3.9 (Open Rule)**.** *Rule r = liveIn(x, y) ← workIn(x, z) is not a* closed rule*, since the variable z appears only once in r. r is an open rule.*

Definition 7 (Non-redundant Rule)**.** *Given a knowledge base $\mathcal{K}$ and a rule $r = H \leftarrow B_1 \wedge B_2 \wedge \dots B_n$, r is a non-redundant rule if no atom in r is entailed by other atoms in r with respect to $\mathcal{K}$, i.e., if, $\forall i \in \{0, 1, \dots, n\}$, with $B_0 = H$, results: $\bigwedge_{j \neq i} B_j \not\models_{\mathcal{K}} B_i$,*

The constraint to non-redundant rule avoids mining rules that are obvious.

Example 3.10. *Given a knowledge base $\mathcal{K}$ with the following TBox:*

$$\begin{aligned}\mathcal{T} = \{\ & \mathit{AnimalDoctor} \sqsubseteq \mathit{Human},\\ & \mathit{Male} \sqsubseteq \mathit{Human},\\ & \mathit{Dog} \sqsubseteq \mathit{Animal},\\ & \exists.\mathit{hasPatient}.\top \sqsubseteq \mathit{AnimalDoctor},\\ & \top \sqsubseteq \forall \mathit{hasPatient}.\mathit{Animal}\ \}\end{aligned}$$

- *A trivial rule:*
  - *Human(x) ← hasPatient(x, y) ∧ Human(x) ∧ Dog(y)*
- *Redundant rules:*
  - *Rule Human(x) ← hasPatient(x, y) ∧ Dog(y) is a redundant rule, since the atom Human(x) is entailed by the domain of hasPatient (the domain of hasPatient is AnimalDoctor, such that AnimalDoctor is subsumed by Human) with respect to $\mathcal{K}$.*

  * *Rule Dog(y) ← hasPatient(x, y) ∧ Animal(y) ∧ Human(x) is also a redundant rule, since the atom Human(x) is entailed by the domain of hasPatient and Animal(y) is entailed by the range of hasPatient as well (the range of hasPatient is Animal) with respect to $\mathcal{K}$.*

– *A non-redundant rule:*

  * *Dog(y) ← hasPatient(x, y) ∧ Male(x)*

3.6 Basic metrics for rules evaluation

In order to discover all frequent and fit hidden patterns, metrics are a necessary tool used to assess the quality of a rule. The following metrics are basic metrics that we adopted for evaluation of the quality of a rule.

Definition 8 (Binding)**.** *A binding is a mapping from a set of variables to a set of individuals that occur in the ABox.*

Given a rule $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$, let us denote:

- $\Sigma_H(r)$ the set of distinct bindings of the variables occurring in the head of r, formally: $\Sigma_H(r) = \{binding\ V(H)\}$
- $E_H(r)$ the set of distinct bindings of the variables occurring in the head of r provided the body and the head of r are satisfied, formally:
  $E_H(r) =$ *{ binding f of V(H) / there is a binding f' of* $V(B_1 \wedge \cdots \wedge B_n)$ *that extends f, such that* $f'(B_1 \wedge \cdots \wedge B_n \wedge H)$ *is satisfied}.*
  Since rules are the safety condition, $V(H) \subseteq V(B_1 \wedge \cdots \wedge B_n)$
- $M_H(r)$ the set of distinct bindings of the variables occurring in the head of r also appearing as binding for the variables occurring in the body of r (since rules are the safety condition), formally:
  $M_H(r) =$ *{ binding f of V(H) / there is a binding f' of* $V(B_1 \wedge \cdots \wedge B_n)$ *that extends f, such that* $f'(B_1 \wedge \cdots \wedge B_n)$ *is satisfied}.*

- $P_H(r)$ the set of distinct bindings of the variables occurring in the head of r provided that the body and the domain variable in the head of r along with

a new variable (which replaces the range variable in the head) are satisfied. Particularly, this applies when a role atom is in the head of the considered rule. Formally:

$P_H(r) = \{$ *binding f of V(H) | there is a binding f' of* $V(B_1 \wedge \cdots \wedge B_n) \cup v_{rng}(H')$ *that extends f, such that* $f'(B_1 \wedge \cdots \wedge B_n \wedge H')$ *is satisfied*$\}$ where

- H and H' are role atoms with the same the predicate symbol R;
- $V(H) \subseteq V(B_1 \wedge \cdots \wedge B_n)$ since rules are safety condition
- $v_{dom}(H) = v_{dom}(H')$ and $v_{rng}(H) \neq v_{rng}(H')$;

with v_{dom} and v_{rng} standing for the domain and range variables respectively of the predicate symbol R

- $v_{rng}(H') \notin V(B_1 \wedge \cdots \wedge B_n)$;

$P_H(r)$ is used to assist the computation of the metric PCA-Confidence (Definition 12). $P_H(r)$ is computed by creating counter-examples. Counter-examples are created as follows: Given a role $r(x, y)$, we assume that if we know one y for a given x and r, then we know all y for that x and r. A new range variable $v_{rng}(H')$ to be created to replace the range variable $v_{rng}(H)$ in the head, is the way to keep positive and negative examples for r.

Following [GTHS13], we redefine a few basic definitions, modified from the classical ones (as given e.g. in [AIS93]) to ensure suit with the SWRL rules.

Definition 9 (Rule Support)**.** *Given a rule* $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$*, its support is the number of distinct bindings of the variables in the head, provided the body and the head of r are satisfied jointly, formally:*

$$\text{supp}(r) = |E_H(r)|. \tag{3.1}$$

Example 3.11 (Computation of Rule Support)**.** *Given the rule* $r = \mathsf{feed}(x, y) \rightarrow \mathsf{love}(x, y)$ *and assuming the bindings in Table 3.3.*

By the Definition 9, we have $\text{supp}(r) = 1$*, as there is just one binding for the rule head (*$\mathsf{feed}(Anna, Dog)$*) allowing the head* $\mathsf{love}(Anna, Dog)$ *and the body* $\mathsf{feed}(Anna, Dog)$ *to be jointly satisfied.*

Definition 10 (Head Coverage for a Rule)**.** *Given the rule* $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$*, its head coverage is the ratio between the rule support and the distinct variable bindings from the head of the rule:*

$$\text{headCoverage}(r) = |E_H(r)|/|\Sigma_H(r)|. \tag{3.2}$$

Table 3.3: A rule and a KB with assertions about two relations between people and pets.

$r = \mathsf{feed}(x, y) \rightarrow \mathsf{love}(x, y)$			
feed		love	
Anna	Dog	Anna	Dog
Anna	Cat	George	Cat
Peter	Pig		

headCoverage(r) *ranges from 0 to 1.*

Example 3.12 (Computation of Head Coverage)**.** *Given the rule r in Table 3.3 and the corresponding bindings,* headCoverage$(r) = \frac{1}{2}$ *since there are two bindings for the head of r:* $\{\mathsf{love}(Anna, Dog), \mathsf{love}(George, Cat)\}$.

Definition 11 (Rule Confidence)**.** *Given a rule $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$, its confidence is defined as the ratio between the rule support and the number of bindings in the rule body:*

$$\text{conf}(r) = |E_H(r)|/|M_H(r)|. \tag{3.3}$$

conf(r) *ranges from 0 to 1.*

Example 3.13 (Computation of Rule Confidence)**.** *Given the rule r in Table 3.3 and the corresponding bindings,* conf$(r) = \frac{1}{3}$*, since there are three bindings,* $\{\mathsf{feed}(Anna, Dog), \mathsf{feed}(Anna, Cat), \mathsf{feed}(Peter, Pig)\}$ *for the body of r.*

Rule Confidence in Definition 11 is a standard confidence metric used in CWA (Closed World Assumption) where all facts neither in the knowledge base nor consequence of the knowledge base are regarded as negative evidences. In CWA, no distinction between *incorrect* predictions, i.e., bindings σ matching r such that $\mathcal{K} \models \neg H\sigma$, and *unknown* predictions, i.e., bindings σ matching r such that both $\mathcal{K} \models H\sigma$ and $\mathcal{K} \models \neg H\sigma$, is made. Ontologies operate under OWA (Open World Assumption). In OWA, a prediction that is not contained in the knowledge base is not necessarily false, it is just *unknown*. Reasoning on ontologies is also grounded on the OWA and our goal is to maximize correct predictions, not just describing the

available data. Hence, we also adopt the PCA Confidence [GTHS13] in Definition 12 that is able to take into account the OWA.

Definition 12 (Rule PCA-Confidence)**.** *Given the rule* $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$*, its PCA (Partial Completeness Assumption) confidence is defined as follows:*

$$\text{pcaconf}(r) = \begin{cases} |E_H(r)|/|M_H(r)|, & \text{if } H \text{ is a concept atom;} \\ |E_H(r)|/|P_H(r)|, & \text{if } H \text{ is a role atom.} \end{cases} \tag{3.4}$$

pcaconf(r) ranges from 0 to 1. For the example described in Table 3.3, pcaconf$(r) = \frac{1}{2}$.

- The head of the rule in Table 3.3 is a role atom.
- $P_H(r)$ consists of two examples for the rule in Table 3.3. love(Anna, Dog) is positive example; love(Anna, Cat) is the prediction and is regarded as negative example, because we already know a different animal for Anna. The prediction love(Peter, Pig) is completely disregarded as evidence, because we donot know what animal Peter loves.

3.7 Extensive metrics for rules evaluation

Besides the metrics we have described so far, in this section, we give some other metrics originally proposed for scoring association rules. There are two kind of metrics, symmetric and asymmetric metrics. In our work, we focus only on introducing asymmetric metrics that are suitable for assessing rules coded in SWRL because the form of SWRL rules having the general form $H \longleftarrow B$ and values of an asymmetric metric for $H \longleftarrow B$ and $B \longleftarrow H$ may not be the same. We do not use symmetric metrics because they are only appropriate for evaluating itemsets, which is values of an symmetric metric for $H \longleftarrow B$ and $B \longleftarrow H$ are the same.

We modified the old definitions with the symbols defined in Section 3.6 and the following symbols to ensure suit with the SWRL rules.

- Σ_i total number of individuals inside a KB.
- Given the rule $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$, N is a number defined as follows:

$$N = \begin{cases} \Sigma_i, & \text{if } H \text{ is a concept atom;} \\ P^2_{\Sigma_i} = \frac{(\Sigma_i)!}{(\Sigma_i - 2)!}, & \text{if } H \text{ is a role atom.} \end{cases}$$

Definition 13 (Laplace for a Rule)**.** *Given a rule* $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$*, its Laplace [CB91], often used to grade rules for classification goals, is defined as*

$$\text{Laplace}(r) = \frac{|E_H(r)| + 1}{|M_H(r)| + 2} \tag{3.5}$$

Laplace(r) ranges from 0 to 1. For the example described in Table 3.3, Laplace$(r) = \frac{2}{5}$.

Definition 14 (Conviction for a Rule)**.** *Given a rule* $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$*, its Conviction [BMUT97], measuring the intensity of implication of a rule, is defined using the confidence metrics in the denominator:*

$$\text{conviction}(r) = \frac{N - |\Sigma_H(r)|}{N(1 - |\text{conf}(r)|)} \tag{3.6}$$

conviction(r) ranges from 0.5 to $+\infty$. Conviction is infinite for confidence 1 and is 1 if the head and the body are independent. Conviction values in (0.5, 1) mean negative dependence and far from 1 indicate interesting rules. For the example described in Table 3.3, conviction$(r) = \frac{7}{5}$.

Definition 15 (Certainty factor for a Rule)**.** *Given a rule* $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$*, its Certainty Factor [FS00] represents uncertainty in the rule and is defined as follows:*

$$\text{cf}(r) = \begin{cases} \frac{\text{conf}(r) - \frac{|\Sigma_H(r)|}{N}}{1 - \frac{|\Sigma_H(r)|}{N}}, & \textit{if } \text{conf}(r) > \frac{|\Sigma_H(r)|}{N} \ ; \\ \frac{\text{conf}(r) - \frac{|\Sigma_H(r)|}{N}}{\frac{|\Sigma_H(r)|}{N}}, & \textit{if } \text{conf}(r) < \frac{|\Sigma_H(r)|}{N} \ ; \\ 0, & \textit{if } \text{conf}(r) = \frac{|\Sigma_H(r)|}{N} \ . \end{cases} \tag{3.7}$$

cf(r) ranges from -1 to +1. For the example described in Table 3.3, cf$(r) =$ 0.286.

Definition 16 (Added value for a Rule)**.** *Given a rule* $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$*, Added Value [SM99] for the rule* r *is defined as:*

$$\text{av}(r) = \text{conf}(r) - \frac{|\Sigma_H(r)|}{N} \tag{3.8}$$

av(r) ranges from -0.5 to 1. This metric is more meaningful when the amount of evidence is large, because it relies on probabilities. For the example described in Table 3.3, av$(r) = 0.267$.

Definition 17 (J-Measure for a Rule). *Given a rule $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$, its J-Measure [SG91] is defined according to the probability distribution of individuals as follows:*

$$J(r) = \frac{|E_H(r)|}{N} \log_2 \frac{N|E_H(r)|}{|M_H(r)||\Sigma_H(r)|} + \frac{|M_H(r)|-|E_H(r)|}{N} \log_2 \frac{N(|M_H(r)|-|E_H(r)|)}{|M_H(r)|(N-|\Sigma_H(r)|)}. \quad (3.9)$$

$J(r)$ ranges from 0 to 1. For the example described in Table 3.3, $J(r) = 0.045$.

Definition 18 (Gini index for a Rule). *Given a rule $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$, its Gini Index [BFOS84] is defined according to the probability distribution of individuals from the sum of squared probabilities as follows:*

$$\begin{aligned} \mathrm{gn}(r) &= \frac{|M_H(r)|}{N}\left[\left(\frac{|E_H(r)|}{|M_H(r)|}\right)^2 + \left(\frac{|M_H(r)|-|E_H(r)|}{|M_H(r)|}\right)^2\right] - \left(\frac{|\Sigma_H(r)|}{N}\right)^2 \\ &+ \frac{N-|M_H(r)|}{N}\left[\left(\frac{|\Sigma_H(r)|-|E_H(r)|}{N-|M_H(r)|}\right)^2 + \left(\frac{(N-|M_H(r)|)-(|\Sigma_H(r)|-|E_H(r)|)}{N-|M_H(r)|}\right)^2\right] \\ &- \left(\frac{N-|\Sigma_H(r)|}{N}\right)^2. \end{aligned}$$

$\mathrm{gn}(r)$ ranges from 0 to 1. For the example described in Table 3.3, $\mathrm{gn}(r) = 0.016$.

3.8 Evaluation of Rule Precision

Definition 19 (Rule Precision). *Given the rule $r = H \leftarrow B_1 \wedge \ldots \wedge B_n$, its precision is the ratio of the number of correct predictions made by r and the total number of correct and incorrect predictions (predictions logically contradicting $\mathcal{K}$), leaving out the predictions with unknown truth value.*

This metric expresses the ability of a rule to perform correct predictions, but it is not able to take into account the induced knowledge, that is the *unknown* predictions. For this reason, the metrics proposed in [FdE08a] are also considered (for the evaluation in Sections 4.4, 5.4 and 6.2):

- *match rate*: number of predicted assertions in agreement with facts in the complete ontology, out of all predictions;
- *commission error rate*: number of predicted assertions contradicting facts in the full ontology, out of all predictions;
- *induction rate*: number of predicted assertions that are not known (i.e., for which there is no information) in the complete ontology, out of all predictions.

In order to compute these metrics, we need three kind of samples of ontology: Stratified sample, Complete sample and Full sample. These samples of ontology is introduced in Section 4.4.1.

Chapter 4

Level-Wise Generate-And-Test

Contents

4.1 Introduction 39
4.2 Downward Refinement Operator 41
4.2.1 Preliminaries 41
4.2.2 Definition of the downward refinement operator 42
4.3 The algorithm 43
4.3.1 Discover all possible frequent patterns 44
4.3.2 Obtain multi-relational association rules 55
4.4 Experiments and evaluation 55
4.4.1 Experimental protocol 55
4.4.2 Experimental evaluation 59
4.5 Summary 62

4.1 Introduction

In this chapter, we present an algorithm that aims at discovering hidden knowledge patterns in the form of multi-relational association rules coded in SWRL [+04]. This algorithm uses an ILP (Inductive Logic Programming) approach to induce rules from facts. The goal of the algorithm is to explore all possible rules in a given space, specifically, this space is determined by the maximum length of the rules to be discovered (Figure. 4.1).

An overview of the algorithm is shown in Figure. 4.2. During processing, the algorithm generates connected rules by applying a downward refinement operator (Section 4.2) on all previously generated rules whose length is less than a given maximum length. The rules not satisfying a certain condition will be pruned, while

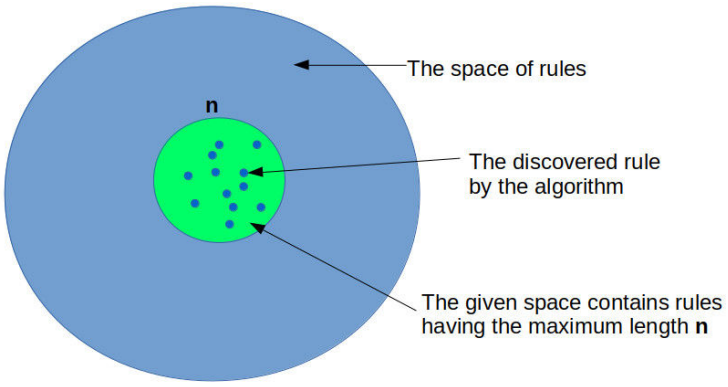


Figure 4.1: The space of rules

the rules passing all conditions will be added into a set of discovered rules, which contains the final results of the algorithm.

Basically, discovered rules must pass three verification steps:

1. Use refinement operator to generate rules satisfying the language bias: This step ensures that rules to be generated are neither redundant nor trivial.

2. Use metrics to assess the quality of the rules: We regard passing a minimum threshold value as a basic quality assurance for a rule.

3. Checking if the rules are consistent with the knowledge base: Each rule is considered separately with the ontology in order to check whether it contradicts the knowledge base.

Besides exploring all discovered rules in a given space, another advantage of this algorithm is that we never have to consider and evaluate redundant or trivial rules because they are never generated. However, the limitation of this method is that it is difficult to generate long rules, thus this algorithm is only suitable for discovering short rules.

The rest of this chapter is structured as follows: In Section 4.2, we propose a downward refinement operator used by the algorithm to traverse the space of rules and find valid rules. Details of the algorithm are presented in Section 4.3 while its experimental evaluation is given in Section 4.4. Conclusions are drawn in Section 4.5.

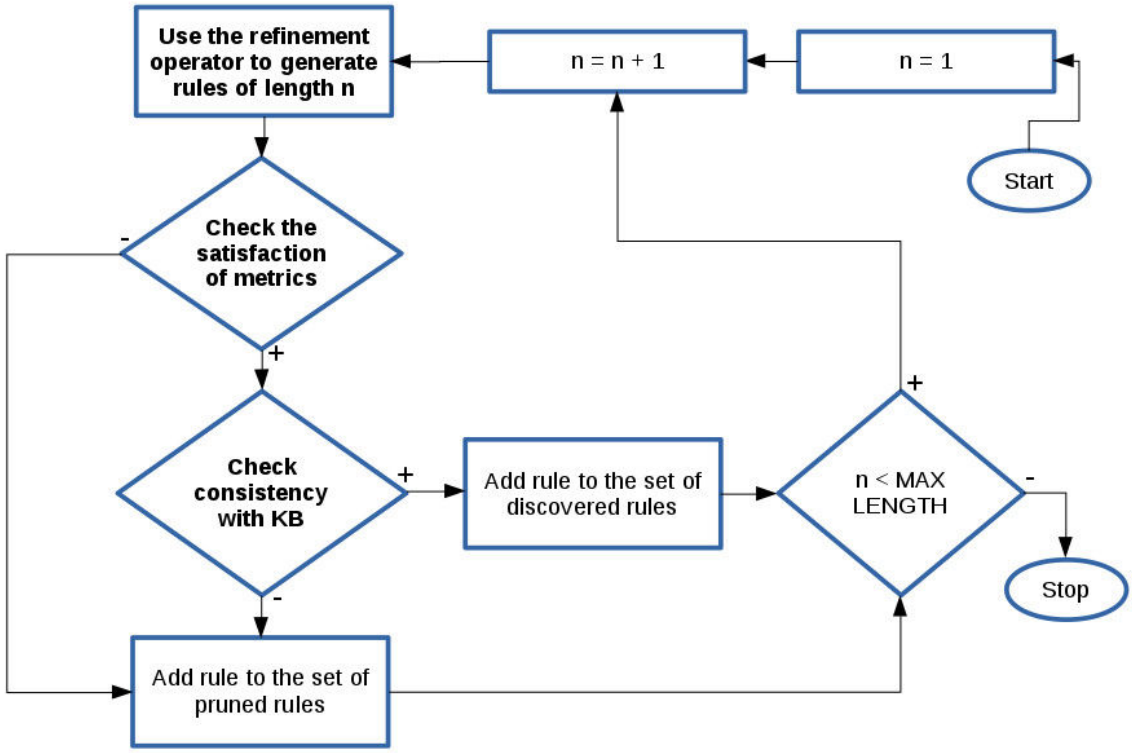


Figure 4.2: Overview of the algorithm

4.2 Downward Refinement Operator

4.2.1 Preliminaries

We regard deriving rules from facts as a search process in the space of rules. In order to perform this operation, we use an idea from ILP [NCW97], which offers an ordering on search space and uses a refinement operator to traverse it and find hypotheses. In this algorithm, we shall deal with a downward refinement operator constructing specializations of rules.

Definition 20 (Downward refinement operator [LH07])**.** *A quasi-ordering is a reflexive and transitive relation. Let S be a set and $\preceq$ a quasi-ordering on S. In the quasi-ordered space (S, $\preceq$) a downward refinement operator ρ is a mapping from S to 2^S, such that for any $C \in S$ we have that $C' \in \rho(C)$ implies $C' \preceq C$. C' is called a specialization of C.*

Suppose we call $E_H(r)$ a set of distinct bindings of the variables in the head of r (see Section 3.6). We rely on Definition 20 to come up with an idea of searching

in the space of rules. A rule r covers a rule r' ($r \succ r'$), which means that $E_H(r') \subseteq E_H(r)$. This provides a suitable order for searching the space of rules.

Definition 21 (Types of operators)**.** *Let $\Re$ be a set of rules to be formed in SWRL (Section 3.4). A downward refinement operator ρ is a mapping from $E_H(r)$ ($r \in \Re$) to its powerset. ρ is called*

- ***finite*** *iff $\rho(E_H(r))$ is finite for any rule r.*
- ***proper*** *iff for all rules r and r', $E_H(r') \in \rho(E_H(r))$ implies $r' \not\equiv r$.*
- ***complete*** *iff for all rules r and r' with $r \succ r'$ we can reach a rule r'' from r by ρ with $r' \equiv r''$.*
- ***weakly complete*** *iff for all rules r with $E_H(r) \preceq E_H(\Re)$ we can reach a rule r' from $E_H(\Re)$ by ρ with $r \equiv r'$.*
- ***ideal*** *iff it is finite, complete, and proper.*

4.2.2 Definition of the downward refinement operator

4.2.2.1 Definition of the operator

Assume

- $\Re$ is a set of rules to be formed in SWRL (r: $H \leftarrow B_1 \wedge ... \wedge B_n$)
- $E_H(\Re)$ is a set of distinct bindings of the variables occurring in the head of the rules in $\Re$.
- A_f is a list containing atomic concept names and atomic role names whose cardinality of instances is higher than a given threshold θ_f.

Definition 22. *The downward refinement operator $\rho : E_H(\Re) \longrightarrow \wp(E_H(\Re))$ is defined as:*

$$\rho(E_H(r)) := \{binding\ V(H) \mid \exists binding\ V(B_1 \wedge ... \wedge B_n \wedge A) : B_1 \wedge ... \wedge B_n \wedge A \wedge H\}$$

For $r \in \Re$; $\wp(E_H(r))$ is the powerset of distinct bindings of the variables in the head of r; $A \in A_f$; $V(X)$ denotes the set of all the variables occurring in X.

Example 4.1. *Suppose we have:*
$A_f = \{fatherOf, grandfatherOf, sonOf, Male\}$
r: fatherOf(x, y) $\leftarrow$ *grandfatherOf(x, z)*
Distinct bindings of the variables in the head of each following rules are contained in $\wp(E_H(r))$*. These rules are generated by using mapping* ρ *with respect to r:*

- *fatherOf(x, y)* $\leftarrow$ *grandfatherOf(x, z)* $\wedge$ *sonOf(z, y)*
- *fatherOf(x, y)* $\leftarrow$ *grandfatherOf(x, z)* $\wedge$ *Male(z)*
- *...*

4.2.2.2 An ideal downward refinement operator

Proposition 1. *ρ is a finite and complete downward refinement operator on space $(E_H(\Re), \preceq)$*

Proof:

- It is easily noticed that ρ is a downward refinement operator. ρ adds an atom to a rule r by using $\wedge$, thus $\rho(E_H(r)) \subseteq E_H(r)$. Hence, $E_H(r') \preceq E_H(r)$ for all $E_H(r') \in \rho(E_H(r))$.
- $\rho(E_H(r))$ is finite because $\wp(E_H(r))$ consists only of rules of length at most length(r) + 1, $\rho(E_H(r)) \subseteq E_H(r)$ and $E_H(r)$ is finite.
- $\rho(E_H(r))$ is complete because for all r and r' with $E_H(r') \preceq E_H(r)$, we can find an atom A $\in A_f$ in order to generate a new rule r'', such that $E_H(r') \equiv E_H(r'')$.

We notice that $\rho(E_H(r))$ is not proper because, for all r and r' with $E_H(r') \preceq E_H(r)$, we can find an atom A $\in A_f$ in order to generate a new rule r'', such that $E_H(r') \equiv E_H(r'')$.

4.3 The algorithm

The algorithm is divided into two main phases. The former is to discover all possible frequent patterns by implementing the level-wise generate-and-test approach. The

latter is to obtain multi-relational association rules from the discovered frequent patterns by considering the first atom in the pattern as the head and the remaining atoms as the body.

4.3.1 Discover all possible frequent patterns

The level-wise generate-and-test approach is applied to discover all possible frequent patterns having a given maximum length. Firstly, a list of general patterns is initialized and each of them contains a single atom which satisfies a given criterion. Next, a level-wise approach is applied to discover the frequent patterns by using relevant operators to specialize the pattern continuously until a certain stopping criterion is met. Specifically, discovery activities on each level are divided into two phases: the generate phrase creates a set of specialized patterns and then the test phrase evaluates each pattern in the set of patterns for possible pruning.

Algorithm 1 describes the whole process of discovery of all possible frequent patterns that have the maximum length of a given MAX_RULE_LENGTH. The algorithm can be expressed in the following steps:

1. First, the function **CreateGeneralPatterns** is called to generate all general patterns, each consisting of only one single atom referring to a concept name or role name in the knowledge base; the function only maintains all concept and role names whose cardinality extensions, instance retrieval, is higher than a threshold θ_f. All output patterns of the function **CreateGeneralPatterns** are stored in the queue q and we call a set of these patterns a set of frequent atoms.

   Example 4.2. *Give two sets of concepts names N_C, role names N_R and the number of facts of each atom with $\theta_f = 5$*

   $N_C = \{$*Male(50), Female(27), Father(8), Mother(22), Son(2), Daughter(15)*$\}$

   $N_R = \{$*FatherOf(15), MotherOf(18), SonOf(2), DaughterOf(5)* $\}$

   Output patterns of the function ***CreateGeneralPatterns*** *are stored in the queue q as follows:*

   $q = \{$*Male, Female, Father, Mother, Daughter, FatherOf, MotherOf*$\}$

Algorithm 1: Discover multi-relational association rules from a populated ontological KB

input : $\mathcal{K}$: ontological KB; θ_f: frequency threshold;

output: $frequent$: set of frequent patterns discovered from $\mathcal{K}$

```
1  infrequent ← Ø; frequent ← Ø;
2  q ← CreateGeneralPatterns(K, θ_f);
3  while ¬q.isEmpty() do
4      p ← q.dequeue();
5      specPatternList ← GenerateSpecializedPatterns(p);
6      specializationAdded ← false;
7      foreach p' ∈ specPatternList do
8          pruned ← EvaluatePatternForPruning(K, p, p', q, infrequent);
9          if pruned then
10             infrequent ← infrequent ∪ {p'} ;
11         end
12         else if p'.length() < MAX_RULE_LENGTH then
13             q.enqueue(p');
14             specializationAdded ← true;
15         end
16         else if IsSafe(p'.asRule()) then
17             frequent ← frequent ∪ {p'} ;
18             specializationAdded ← true;
19         end
20     end
21     if ¬specializationAdded and p.length() ≥ 2 then
22         p_safe ← GetSafePatternOrAncestorPattern(p);
23         if (p_safe ≠ null) and (p_safe ∉ frequent) then
24             frequent ← frequent ∪ {p_safe} ;
25         end
26     end
27 end
28 return frequent
```

2. Next, the function **GenerateSpecializedPatterns** is called to specialize each general pattern which is dequeued from q until q is empty. This function performs to specialize the general pattern by adding a new atom into the pattern for each level and then evaluating the achieved pattern for possible pruning. Pattern specialization is performed until a stopping criterion is met, precisely here it is the maximum length of the pattern is reached. The following two operators are applied to generate the specialized pattern from a given pattern, namely at each step of the specialization process, the operators are applied to obtain rules satisfying the language bias (Section 3.5):

   (a) Add a concept atom (see Section 4.3.1.1 for more details): This operator adds an atom into a given pattern, this atom is taken from the set of frequent atoms and its variable argument already appears in the given pattern. This atom (concept name) can already appear in the given pattern; in that case, a different variable name must be selected.

   Example 4.3. *Given the pattern:*
   p : [Male(x), siblingOf(y, x)]
   and the concept ***Male*** *is added to p.*
   The variable name of the concept ***Male*** *can only be* ***y*** *because the variable x is in the previous* ***Male*** *concept. Therefore, new pattern after specializing with the concept* ***Male*** *is:*
   p_{new} *: [Male(x), siblingOf(y, x), Male(y)]*

   (b) Add a role atom (see Section 4.3.1.2 for more details): This operator also adds an atom into a given pattern, but this atom is a role name in the set of frequent atoms and at least one of its variable arguments already appears in the given pattern while other can already appear or can be a new variable. This atom (role name) can already appear in the given pattern.

   Example 4.4. *Given the pattern:*
   p : [Male(x), siblingOf(y, x)]
   and the role ***motherOf*** *is added to p.*
   The domain variable name of the role ***motherOf*** *can not be* ***x*** *because* ***x*** *is a* ***Male****. Therefore, new pattern after specializing with the role*

motherOf *can be:*
p^1_{new} *: [Male(x), siblingOf(y, x), motherOf(y, z)] or*
p^2_{new} *: [Male(x), siblingOf(y, x), motherOf(z, x)]*
...

3. The function **EvaluatePatternForPruning** is called to evaluate the pattern for possible pruning at each specialization level. (see Section 4.3.1.3 for more details)

4. A list of the infrequent patterns is used to contain the specialized patterns to be pruned and a list of the frequent patterns is used to contain the discovered frequent patterns.

   (a) The lines 9-10 in Algorithm 1 show that if the specialized pattern p' is pruned, it is added to the list of the infrequent patterns.

   (b) On the contrary, the lines 12-13 in Algorithm 1 show that if the pattern is not pruned and the pattern length does not reach the given maximum length, the specialized pattern p' is enqueued to the queue q and it is considered as a general pattern.

   (c) In the remaining case, the lines 16-17 in Algorithm 1 show that if the pattern is not pruned but the pattern length reaches the given maximum length, the specialized pattern p' is added to the list of the discovered frequent patterns if it satisfies the safety condition (see Definition 4).

   (d) If all specialized patterns of a given pattern are pruned, the lines 21-24 in Algorithm 1 show that a pattern is computed as follows so that it is added to the list of the discovered frequent patterns:

      i. Either that pattern is the given pattern satisfying the safety condition.

      ii. Or that pattern is a first ancestor of the given pattern if the given pattern does not satisfy the safety condition but its first ancestor satisfies.

      iii. And that pattern does not already appear in the list.

5. When q is empty, namely all the general patterns are fully specialized, the list of the discovered frequent patterns is returned (line 28 of Algorithm 1).

4.3.1.1 The specialization operator "Add a concept atom"

Algorithm 2: Implements the specialization operator "add a concept atom" which, given the current pattern r and the candidate concept atom C', returns all possible non-redundant patterns w.r.t. the combination of variables

input : r: the pattern to be specialized; C': the candidate concept atom; $concepts_r$: concept names appearing in the pattern under construction; $roles_r$: role names appearing in the pattern under construction; $vars_r$: variable names appearing in the pattern under construction;

output: *specializedPatternsGivenAConceptAtom*: the list of all non-redundant specializations for the input pattern, given the candidate concept atom C'

1 *specializedPatternsGivenAConceptAtom* $\leftarrow \emptyset$;
2 $subC_r \leftarrow concepts_r$*.getConceptsSubsumedBy(*C'*)*;
3 $superC_r \leftarrow concepts_r$*.getConceptsSubsuming(*$C'$*)*;
4 $subR_r \leftarrow roles_r$*.getRolesWithDomainOrRangeSubsumedBy(*C'*)*;
5 *used* $\leftarrow$ *subC*$_r$*.getVars()* $\cup$ *superC*$_r$*.getVars()* $\cup$ *subR*$_r$*.getVars()*;
6 **foreach** $v \in vars_r \setminus$ *used* **do**
7 | *specializedPatternsGivenAConceptAtom.add(r* $\wedge$ *C'(v))*;
8 **end**
9 **return** *specializedPatternsGivenAConceptAtom*

This operator is used to add an atom being a concept name in the knowledge base into a given pattern (see Algorithm 2 for more details) and the added atom can already appear. After adding the concept name to the pattern, the operator must ensure the obtained rule satisfies the language bias (Section 3.5), and is neither redundancy nor triviality. The lines 2-7 in Algorithm 2 show that non-redundancy and non-triviality are assured by the following check:

1. A candidate concept name is never added as a concept atom if its variable name is the same as the variable name of a concept atom in the given pattern, and that concept atom subsumes or is subsumed by the candidate concept name (lines 2-3 in Algorithm 2).

Example 4.5. *Given the pattern:*
p : [Male(x), siblingOf(y, x)]
and the concept ***Brother*** *is added to p.*
The variable name of the concept ***Brother*** *can only be* ***y*** *because the variable* ***x*** *already appears in the previous* ***Male*** *concept, and* ***Brother*** $\sqsubseteq$ ***Male****.*
Therefore, new pattern after specializing with the concept ***Brother*** *is:*
p_{new} *: [Male(x), siblingOf(y, x), Brother(y)]*

2. A candidate concept name is never added as a concept atom if its variable name already appears in a role atom of the given pattern, whose domain and/or range is subsumed by the candidate concept name (line 4 in Algorithm 2).

   Example 4.6. *Given the pattern:*
   p : [FatherOf(x, y), HusbandOf(x, z)]
   and the concept ***Male*** *is added to p.*
   The variable name of the concept ***Male*** *can not be* ***x*** *because the variable* ***x*** *already appears in the domain of previous* ***FatherOf*** *and* ***HusbandOf*** *roles, and* $\exists$***FatherOf.***$\top \sqsubseteq$ ***Male***, $\exists$***HusbandOf.***$\top \sqsubseteq$ ***Male****. Therefore, new pattern after specializing with the role* ***Male*** *can be:*
   p^1_{new} *: [FatherOf(x, y), HusbandOf(x, z), Male(y)] or*
   p^2_{new} *: [FatherOf(x, y), HusbandOf(x, z), Male(z)]*
   p^2_{new} *is acceptable to generate in this function, however it will be pruned when considered jointly with the ontology and unsatisfiable because the concept* ***Male*** *is disjoint with the range of the role* ***HusbandOf****.*

4.3.1.2 The specialization operator "Add a role atom"

This operator is used to add an atom being a role name in the knowledge base into a given pattern (see Algorithm 3 and Algorithm 4 for more details) and the added atom can already appear. After adding the role name to the pattern, the operator must ensure the obtained rule satisfies the language bias (Section 3.5), and is neither redundancy nor triviality. Non-redundancy and non-triviality are assured by the following check:

1. In case there is a new variable: A candidate role name is never added as a

Algorithm 3: Implements the specialization operator "add a role atom introducing a fresh variable" which, given the current pattern r and the candidate role atom R', returns all possible non-redundant patterns w.r.t. the combination of variables and a fresh variable

input : r: the pattern to be specialized; R': the candidate role atom;
$concepts_r$: concept names appearing in the pattern under construction;
$roles_r$: role names appearing in the pattern under construction;
$vars_r$: variable names appearing in the pattern under construction;

output: $specializedPatternsWithFreshVarGivenARoleAtom$: the list of all non-redundant specializations for the input pattern, given the candidate role atom R'

```
1  specializedPatternsWithFreshVarGivenARoleAtom ← ∅;
2  z ← GetFreshVariable();
3  supConceptsOfDomain_r ← concepts_r.getConceptsSubsuming(dom(R'));
4  supConceptsOfRange_r ← concepts_r.getConceptsSubsuming(range(R'));
5  foreach v ∈ vars_r \ supConceptsOfDomain_r.getVars() do
6  |   specializedPatternsWithFreshVarGivenARoleAtom.add(r ∧ R'_i(v, z));
7  end
8  foreach v ∈ vars_r \ supConceptsOfRange_r.getVars() do
9  |   specializedPatternsWithFreshVarGivenARoleAtom.add(r ∧ R'_i(z, v));
10 end
11 return specializedPatternsWithFreshVarGivenARoleAtom
```

Algorithm 4: Implements the specialization operator "add a role atom with all variables bound" which, given the current pattern r and the candidate role atom R', returns all possible non-redundant patterns w.r.t. the combination of variables

input : r: the pattern to be specialized; R': the candidate role atom; $concepts_r$: concept names appearing in the pattern under construction; $roles_r$: role names appearing in the pattern under construction; $vars_r$: variable names appearing in the pattern under construction;

output: $specializedPatternsWithBoundVars$: the list of all non-redundant specializations for the input pattern, given the candidate role atom R'

```
1  specializedPatternsWithBoundVars ← ∅;
2  supConceptsOfDomain_r ← concepts_r.getConceptsSubsuming(dom(R'));
3  supConceptsOfRange_r ← concepts_r.getConceptsSubsuming(range(R'));
4  if R' ∉ roles_r then
5      foreach v ∈ vars_r \ supConceptsOfDomain_r.getVars() do
6          foreach w ∈ vars_r \ supConceptsOfRange_r.getVars() do
7              specializedPatternsWithBoundVars.add(r ∧ R'(v, w));
8          end
9      end
10 end
11 else
12     usedDomVars ← roles_r.getElement(R').getListOfVarsForDomain();
13     usedRangeVars ← roles_r.getElement(R').getListOfVarsForRange();
14     usedDomVars ← usedDomVars ∪ supConceptsOfDomain_r.getVars();
15     usedRangeVars ← usedRangeVars ∪
         supConceptsOfRange_r.getVars();
16     foreach v ∈ vars_r \ usedDomVars do
17         foreach w ∈ vars_r \ usedRangeVars do
18             specializedPatternsWithBoundVars.add(r ∧ R'(v, w));
19         end
20     end
21 end
22 return specializedPatternsWithBoundVars
```

role atom, if the domain (range) of that candidate has a variable name being a new variable and the other variable appears in a concept atom of the given pattern, and that concept atom subsumes the domain (range) of the candidate role name. (Algorithm 3)

Example 4.7. *Given the pattern:*
p : [Male(x), siblingOf(y, x)]
and the role ***fatherOf*** *is added to p.*
The domain variable name of the role ***fatherOf*** *can not be* $\boldsymbol{x}$ *because* $\boldsymbol{x}$ *is a* ***Male*** *and* $\exists$***FatherOf.***$\top \sqsubseteq$ ***Male****. Therefore, new pattern after specializing with the role* ***fatherOf*** *can be:*
p^1_{new} *: [Male(x), siblingOf(y, x), fatherOf(y, z)] or*
p^2_{new} *: [Male(x), siblingOf(y, x), fatherOf(z, y)] or*
p^3_{new} *: [Male(x), siblingOf(y, x), fatherOf(z, x)]*

2. In case there is no new variable: A candidate role name is never added as a role atom, if all variable names in the domain and range of that candidate appear in concept atoms whose predicate symbol subsumes the domain and range of the role name, respectively. (Algorithm 4)

Example 4.8. *Given the pattern:*
p : [Male(x), siblingOf(y, x)]
and the role ***fatherOf*** *is added to p.*
The domain variable name of the role ***fatherOf*** *can not be* $\boldsymbol{x}$ *because* $\boldsymbol{x}$ *is a* ***Male*** *and* $\exists$***FatherOf.***$\top \sqsubseteq$ ***Male****. Therefore, new pattern after specializing with the role* ***fatherOf*** *can be:*
p_{new} *: [Male(x), siblingOf(y, x), fatherOf(y, x)]*

4.3.1.3 On evaluating pattern pruning

Algorithm 5 is used to check for several different pruning conditions on a given specialized pattern. The pattern is pruned, if it satisfies one of the following conditions:

1. The rule obtained from the specialized pattern contradicts the reference knowledge base. The pattern can be pruned if its rule is considered jointly

Algorithm 5: Determine if a pattern has to be pruned or not.

input : $\mathcal{K}$: ontological KB; p: parent pattern of the pattern to be evaluated; p': pattern to be evaluated for pruning; q: list of the generated patterns; $infrequent$: set of the patterns that have been pruned;

output: *pruned*: true if the pattern has to be pruned, false otherwise

```
r' ← p'.asRule(); r ← p.asRule();
if K ∪ r' ⊨ ⊥ then
    return true
end
else if headCoverage(r') < θ_hc then
    return true
end
else if conf(r') - conf(r) < θ_ic then
    return true
end
else if isPatternAlreadyGenerated(p', q) then
    return true
end
else
    foreach infPatt ∈ infrequent do
        if r'.getSupportExtention() ⊆ infPatt.getSupportExtention() then
            return true
        end
    end
end
return false
```

with the ontology (line 2 in Algorithm 5) and unsatisfiable. According to the remark of [JLL10b], the satisfiability check is helpful if disjointness axioms exist in the ontology; this check can be omitted in order to save computational efforts if no disjointness axioms occur.
This condition cannot be satisfied if the ontological knowledge base is consistent and noise-free. However, this method is built to be able to apply to noisy ontologies, thus an unsatisfiable rule can be discovered when considered jointly with the ontology, particularly if low frequency and Head Coverage thresholds (see Definitions 2 and 10) are selected.

2. Head Coverage threshold of the pattern is less than a given threshold θ_{hc} (line 6 in Algorithm 5). This condition ensures that a satisfiable rule to be discovered does not depend on the absolute number of predictions in the rule to be obtained from the pattern.

3. In this algorithm, we consider that the specialized pattern is not improved if it does not contain new information. This means that the pattern is pruned if its confidence does not improve compared to the confidence of its parent (line 8 in Algorithm 5).

4. The function **isPatternAlreadyGenerated** is called to avoid considering the same pattern more than once (line 11 in Algorithm 5) in order to save computational costs. The candidate pattern is pruned if it is the same as an already generated pattern.
We regard two patterns being semantically equivalent as the same; more specifically, if the support extension of the rules that are obtained from the patterns is the same then these patterns have the same semantics. In the function, we use a heuristic to check for an already generated pattern (the same semantics). Because comparing the support extension of the rule to be obtained from the candidate pattern with those of all already generated patterns has a high computational cost, the heuristic selects the patterns whose corresponding rules have the same head coverage value as the candidate pattern generated in advance, then it compares the support extension of the rules to be obtained from these patterns with the candidate pattern. The candidate pattern is pruned if found.

Table 4.1: Key facts about the ontological KBs used.

Ontology	# Concepts	# Roles	# Indiv.	# Declared Assertions	# Decl.+Derived Assertions
Financial	59	16	1000	3359	3814
BioPAX	40	33	323	904	1671
NTMerged	47	27	695	4161	6863

5. A candidate pattern is pruned if its semantics is included in the semantics of previous patterns to be pruned. We use a heuristic to solve this case: the candidate pattern whose corresponding rule has the support extension to be contained in the support extension of a rule which obtained from the pattern in the list of the infrequent patterns. (lines 15-16 in Algorithm 5).

4.3.2 Obtain multi-relational association rules

For each pattern, in order to obtain multi-relational association rules coded in SWRL, the first atom is considered as the head of the rule and the remaining as the rule body.

Example 4.9. *Given the pattern:*
p : [Male(x), siblingOf(y, x), Brother(y)]
The rule corresponding to the pattern p is:
$r : Male(x) \leftarrow siblingOf(y, x) \wedge Brother(y)$

4.4 Experiments and evaluation

4.4.1 Experimental protocol

We carry out testing of the level-wise generate-and-test approach on the following publicly available ontologies (details on them are reported in Table 4.1):

– Financial Ontology (Financial): Describing the banking domain.
http://www.cs.put.poznan.pl/alawrynowicz/financial.owl

– Biological Pathways Exchange (BioPAX): Describing biological pathway data.
http://www.biopax.org/release/biopax-level2.owl

– New Testament Names Ontology (NTNMerged): Describing named entities (people, places, and other classes) in the New Testament, as well as their attributes and relationships.
http://www.semanticbible.com/ntn/ntn-view.html

This experiment aims at the following two basic goals:

1. The first goal of our experiments consisted in assessing the ability of the discovered rules to predict new assertional knowledge for a considered ontological knowledge base.

2. The second goal of our experiments consisted in showing the importance and the value added of exploiting the terminological knowledge and the reasoning capabilities when extracting rules from ontological knowledge bases.

For the first purpose, we use three kinds of samples for each ontology in Table 4.1 in order to compute metrics in Section 3.8. Specific samples are as follows:

1. Stratified Sample: This sample is built by randomly removing $p\%$ of concept assertions, according to Algorithm 6. In the Algorithm 6, in addition to deleting the concept assertions, we also eliminate role assertions or assertional axioms that involve in the removed concept assertions. We perform this so that the removed concept assertions are not entailed by the rest axioms in the ontology.

2. Complete Sample: This sample is an ontology taken from the Table 4.1 and used to create stratified sample.

3. Full Sample: This sample is built by integrating a rule discovered by the stratified sample after running Algorithm 1 into the complete sample. Therefore, the number of full samples is always equal to the number of discovered rules by the stratified sample.

We collected all predictions of the full sample, that is the head atoms of the instantiated rules. All predictions already contained in the stratified sample have been discarded while the remaining predicted facts have been considered. A prediction is assessed as *correct* if it is contained in the complete sample or entailed by

Algorithm 6: The createStratifiedOntology()

input : O_f: an ontology; p: rate of concept assertions is deleted;

output: O_s: a stratified ontology;

```
1  O_s ← O_f;
2  foreach concept in O_s do
3      nIndividuals ← the number of individuals in concept;
4      if nIndividuals > 0 then
5          n ← p * nIndividuals;
6          for i = 1 ⟶ n do
7              Pick an individual in the set of individuals of concept at random;
8              Eliminate individual out of the set of individuals of concept in O_s;
9              Delete role assertions in O_s that their subject or object contains
                individual;
10             Eliminate all other assertional axioms in O_s that contains
                individual;
11         end
12     end
13 end
14 return O_s
```

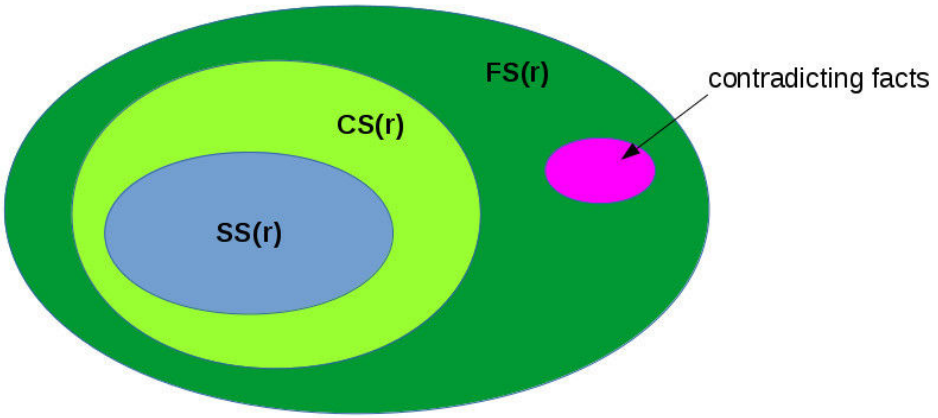


Figure 4.3: The space of rules

the full sample. A prediction is assessed as *incorrect* if it is inconsistent with the full sample.

The Venn diagram in Figure. 4.3 describes sets of facts or predictions, that are in the head atoms of the discovered rule r: SS is the set of facts of the stratified sample; CS is the set of facts of the complete sample; and $FS(r)$ is the set of predictions of the full sample after integrating a rule r into the complete sample. The metrics defined in Section 3.8 are calculated as follows:

- $predictions$ = |FS(r) / SS(r)|: The total of predictions.
- $match\ rate = \frac{|CS(r)/SS(r)|}{predictions}$
- $commission\ error\ rate = \frac{contradicting\ facts(r)}{predictions}$
- $induction\ rate = \frac{|FS(r)/CS(r)|}{predictions}$

match rate shows that the discovered rule is able to predict new assertional knowledge for the considered ontology; *commission error rate* determines whether conflicting knowledge exists in the ontology, if *commission error rate* is greater than 0 then there exists, otherwise not. Facts are regarded as contradictory knowledge when considered jointly with the ontology and unsatisfiable; *induction rate* to be greater than 0 demonstrates the ability to induce new knowledge that is not logically derivable.

For the second purpose, we compared our system with AMIE [GTHS13], which represents the state-of-the-art system in the considered setting, but it is not

Table 4.2: Average performance metrics on each ontology

Ontology	Sample	Match Rate	Comm. Rate	Ind. Rate	Precision	Tot. nr. Predictions
Financial	20%	0.81	0	0.19	1.0	947
	30%	0.81	0	0.19	1.0	1,890
	40%	0.82	0	0.18	1.0	2,960
BioPAX	20%	1.0	0	0	1.0	669
	30%	1.0	0	0	1.0	1,059
	40%	1.0	0	0	1.0	1,618
NTNMerged	20%	0.94	0	0.06	1.0	9,085
	30%	0.9	0	0.1	1.0	9,765
	40%	0.94	0	0.06	1.0	10,418

able to exploit neither terminological information nor reasoning capabilities. Since one of the AMIE key points (as argued in [GTHS13]) is its ability to outperform state-of-the-art ILP systems in terms of number of discovered rules, we compared the number of rules discovered by our system with the number of rules discovered by AMIE, using the same samples of the ontologies in Table 4.1.

4.4.2 Experimental evaluation

For each ontology in Table 4.1, we build three stratified samples by randomly removing respectively 20%, 30%, 40% of the concept assertions, according to Algorithm 6, thus the total number of stratified samples to be created is 9. We ran by repeating 10 times the sampling procedure for each stratified sample and using the following parameters setting:

$$\begin{aligned} \text{MAX\_RULE\_LENGTH} &= 3; & \theta_f &= 1; \\ \theta_{hc} &= 0.01; & \theta_{ic} &= 0.001 \end{aligned}$$

4.4.2.1 The ability to predict assertional knowledge

Looking at Table 4.2, we have some remarks as follows:

1. It is possible to see that very high values of *match rate* are reached for the considered ontologies. This proves that the discovered rules are actually able to predict new assertional knowledge for the considered ontologies.

Table 4.3: Comparison # extracted rules: AMIE vs LW-GAT.

Ontology	Sample	# Rules		Top		
		LW-GAT	AMIE	n	# Predictions LW-GAT	# Predictions AMIE
Financial	20%	177	2	2	29	208
	30%	181	2	2	57	197
	40%	180	2	2	85	184
BioPAX	20%	298	8	8	25	2
	30%	283	8	8	34	2
	40%	272	0	8	50	0
NTNMerged	20%	243	1,129	10	620	420
	30%	225	1,022	10	623	281
	40%	239	1,063	10	625	332

2. Values of *commission error rate* are equal to 0, this means that no contradicting knowledge is predicted. This is one of the expectations of the system because of the exploitation of the terminology and the reasoning capabilities.

3. Values of *induction rate* are greater than 0 for the case of ontologies where cases of concepts and roles for which a large number of assertions is available (Financial and NTNMerged) while for other concepts and roles, few assertions are available (BioPAX). This means that the ability of the level-wise generate-and-test method inducing new knowledge that is not logically derivable, is suitable with ontologies containing a large number of assertions.

4. Values of precision are always the highest one since the induced assertions are not considered for the computation of this metric and no mistake (*commission error rate* is equal to 0) is made.

5. It is also interesting to note how the number of predicted assertions increases when less knowledge is available, since a higher number of assertions have been removed from the ontologies.

4.4.2.2 The importance and the value added

We call the level-wise generate-and-test method LW-GAT. Averaged results are reported in Table 4.3 where it is possible to see that LW-GAT outperformed the

number of rules discovered by AMIE for the case of Financial and BioPAX ontologies. Indeed, the output rules of LW-GAT is open rules (see Example 3.9) and having both concept and role atoms in the head, while AMIE can only output closed rules (Definition 6) with role atoms in the head. Additionally LW-GAT is able to prune redundant rules and rules that are inconsistent when considered jointly with the reference ontology. This is the reason why AMIE registered a larger number of rules then our system for the case of NTNMerged. LW-GAT outperformed AMIE also in terms of number of predictions.

We compared the top n rules, ranked with respect to match rate. n was set equal to the number of rules discovered by AMIE when few rules were discovered, and equal to 10 for the other cases. LW-GAT clearly outperformed AMIE for the cases of BioPAX and NTNMerged. The same did not happen for Financial ontology. This is because outputs of LW-GAT as much as possible specific rules. For the case of Financial ontology, AMIE output two rules, each one having just one atom in the body, while output of LW-GAT several refinements of such two rules, thus preventing the predictions just coming from the general rules. This suggested an improvement of LW-GAT consisting in assessing whether more general or refined rules have to be returned.

Example 4.10. *Two output rules of AMIE in case of Financial ontology are the following:*

- $hasOwner(x,\ y) \leftarrow isOwnerOf(y,\ x)$
- $isOwnerOf(x,\ y) \leftarrow hasOwner(y,\ x)$

While output rules of LW-GAT to be specialization of AMIE output rules are:

- $hasOwner(x,\ y) \leftarrow isOwnerOf(y,\ x) \wedge Woman(y)$
- $hasOwner(x,\ y) \leftarrow isOwnerOf(y,\ x) \wedge Man(y)$
- $hasOwner(x,\ y) \leftarrow isOwnerOf(y,\ x) \wedge isCreditCardOf(z,\ y)$
- $hasOwner(x,\ y) \leftarrow isOwnerOf(y,\ x) \wedge hasOwner(z,\ y)$
- $isOwnerOf(x,\ y) \leftarrow hasOwner(y,\ x) \wedge isUserOf(z,\ y)$
- $isOwnerOf(x,\ y) \leftarrow hasOwner(y,\ x) \wedge isOwnerOf(z,\ y)$
- $isOwnerOf(x,\ y) \leftarrow hasOwner(y,\ x) \wedge IsLoanOf(z,\ y)$

4.5 Summary

In this chapter, we presented a method for discovering multi-relational association rules from ontological knowledge bases, to be used for enriching assertional knowledge. The algorithm obtains all possible rules in a given space and eliminates redundant/trivial rules at the same time. Discovered rules are carefully selected by three basic steps: (i) The generated rules ensure the language bias to be respected. (ii) To pass the evaluation of the metrics. (iii) Satisfaction when considered jointly with the ontology.

Discovered rules are represented in SWRL, which can be easily integrated in the ontology enriching its expressive power and increasing the assertional knowledge that can be derived. In addition, discovered rules may suggest new axioms to be added to the ontology,such as transitivity and symmetry of a role, and/or concept/role inclusion axioms.

The proposed approach has been experimentally evaluated through its application to publicly available ontologies and comparisons with the state of the art system.

Chapter 5

Genetic Algorithm

Contents

5.1 Introduction 63
5.2 Genetic Algorithm 66
5.2.1 Introduction 66
5.2.2 Terminology 67
5.2.3 Structure 67
5.2.4 Steady-state genetic algorithm 68
5.3 The algorithm 69
5.3.1 Representation 69
5.3.2 Operators 72
5.3.3 Fitness function 79
5.3.4 Consistency check 79
5.3.5 Language bias 80
5.4 Experiments and evaluation 80
5.4.1 The ability to predict assertional knowledge 82
5.4.2 The importance and the value added 83
5.5 Summary 86

5.1 Introduction

Similar to the level-wise generate-and-test algorithm, in this chapter we also propose an algorithm for discovering frequent and accurate hidden patterns in the form of multi-relational association rules to be exploited for making predictions of new assertions in the knowledge base. Multi-relational association rules are DL-Safe and expressed in SWRL which can be easily integrated in the ontology, enriching

its expressive power and increasing the assertional knowledge that can be derived. The algorithm that we propose is based on a genetic algorithm.

Although the goal of both algorithms is the same, the genetic algorithm is an improvement of the level-wise generate-and-test algorithm. As we have seen in the previous chapter, the goal of the level-wise generate-and-test algorithm is to retrieve all possible rules determined by the given maximum length. However, this method has a fundamental disadvantage: since the size of the search space increases exponentially with the maximum rule length, it is difficult to obtain long rules because the execution time explodes.

We create the genetic algorithm for rule discovery to overcome the above disadvantage of the level-wise generate-and-test algorithm. Although the genetic algorithm can hardly find all possible rules in the space determined by the given maximum length, it can discover rules of long length along with selecting and keeping the best rules that it traverses in the search space. The weakness of the genetic algorithm is that it can miss some rules (we call these rules the undiscovered rules). To overcome this weakness, we choose the appropriate number of generations for the genetic algorithm to minimize the number of undiscovered rules (Figure. 5.1).

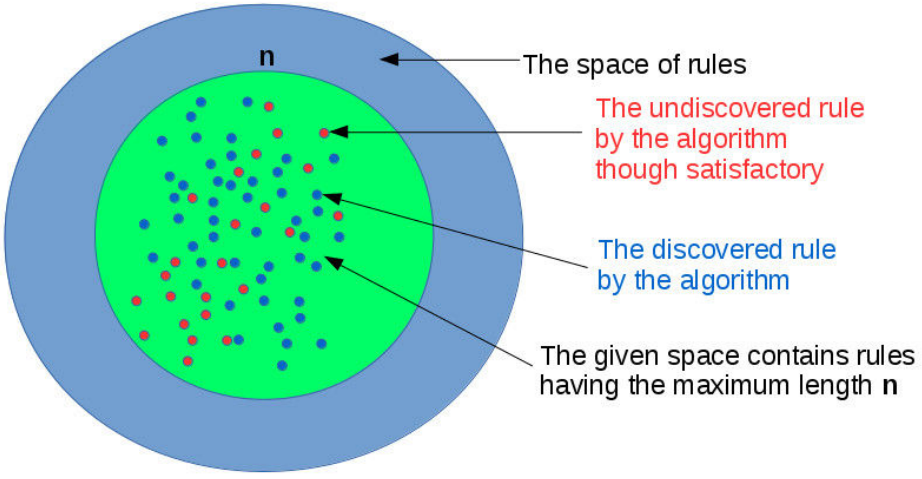


Figure 5.1: The search space of rules used in genetic algorithm

An overview of the algorithm is shown in Figure. 5.2. The algorithm maintains a population of individuals (the patterns) and makes it evolve by iteratively applying a number of genetic operators. A fitness function based on metrics of pattern is

used to assist the genetic operators to select the best individuals. Each individual is generated or produced in a space to be limited in a given maximum length of pattern and compliant with the fixed language bias (Section 3.5).

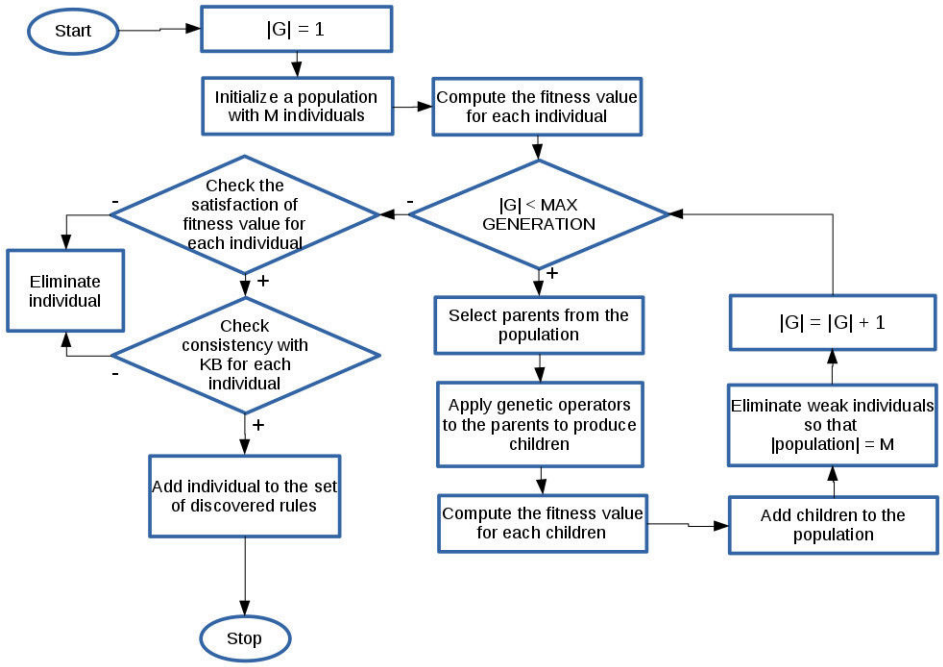


Figure 5.2: Overview of the genetic algorithm

We regard this genetic algorithm as steady-state, this means that children are created by applying genetic operators on selected parents, and then the children are added back into the population to compete with individuals in the old population in order to allow transition into the new population at the next cycle. With this method, we always ensure that the best individuals are retained until the final generation, and they are also the best ones during the process of traversing the search space of the algorithm.

At the final generation, an individual is added to a set of discovered rules if it satisfies the following conditions:

1. The fitness value is higher than a given threshold θ_{fit}.
2. The individual does not contradict the knowledge base. This means that it is considered jointly with the ontology and satisfiable.

The rest of this chapter is structured as follows: In Section 5.2, we briefly intro-

duce the genetic algorithm and steady-state genetic algorithm used in the proposed method. Details of the algorithm are presented in Section 5.3 while its experimental evaluation is given in Section 5.4. Conclusions are drawn in Section 5.5.

5.2 Genetic Algorithm

5.2.1 Introduction

Genetic algorithms ([Hol75], [Gol89], [DeJ02], [ES03]) are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. They are frequently used to find optimal or near-optimal solutions to difficult problems which otherwise would take a long time to solve. Although we are often not able to guarantee that our genetic algorithm can find the global optimum solution to our problem, we can in general obtain a close approximation of the optimal solution.

In the genetic algorithm, a population consisting of individuals is maintained within the search space. Each individual describes a possible solution of a given problem with an associated fitness value representing the competitiveness of an individual. New generations of the population are created by applying genetic operators (selection, crossover, mutation) on individuals of a previous generation. We aim at the creation of genetic operators to produce offspring better than their parents by investigating information from the chromosomes. Actually, we are trying to create new solutions which are better than old solutions and replace the old solutions by the new ones. In this way, we hope that better solutions in successive generations are retained for further development while the least fit solutions will be eliminated. Eventually, the genetic algorithm has three basic stopping criteria as follows:

1. The process is executed iteratively through generations and stops when meeting the optimal solution.

2. As soon as no improvement of the fitness value for the best solution is observed for a fixed number of iterations. In such event, the current best solution is returned as the optimal solution for the problem.

3. The process is executed iteratively until the final generation. The best solution in the final generation is regarded as the optimal solution of problem.

5.2.2 Terminology

- **Population** is a subset of all the encoded solutions to the given problem.
- **Chromosome** is a specific solution to the given problem.
- **Genotype** is the encoding of a solution in the computation space in which the solutions are represented in a way that can be easily understood and manipulated using a computing system.
- **Phenotype** is a solution in the actual real world solution space which is represented in a way close to what is expected in real world situations.
- **Fitness Function** is used to evaluate the quality of the solution to the problem in the population. It evaluates how good a single solution is in a population.
- **Genetic Operators** are used to guide the algorithm towards a solution to a given problem. There are three main types of operators (selection, crossover and mutation), which must work in conjunction with one another in order for the algorithm to be successful.

5.2.3 Structure

We start by randomly initializing a population, determine fitness for all the individuals in the population and then select parents from this population for mating. We then apply crossover and mutation operators on the parents to generate new offspring. Finally, these offspring replace the existing individuals in the population (generational genetic algorithm) or directly compete with their parents to obtain the best individuals and replace the existing parents in the population (steady-state genetic algorithm). This process repeats until it meets a stopping criterion (Figure. 5.3).

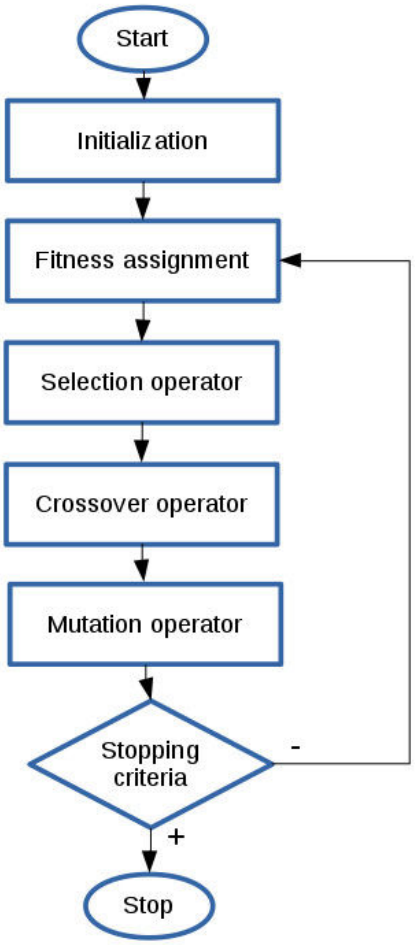


Figure 5.3: Structure of the genetic algorithm

5.2.4 Steady-state genetic algorithm

Actually, steady-state genetic algorithm means that there are no generations. It is different from the generational genetic algorithm in that tournament selection does not replace the selected individuals in the population, and instead of adding the children of the selected parents into the next generation, the two best individuals out of the two parents and two children are added back into the population so that the population size remains constant.

5.3 The algorithm

The genetic algorithm we propose is based on steady-state, however, instead of giving two offspring in direct competition with their parents, we add them back into the population so that they compete with individuals in the whole population. With this change, the best individuals are always retained until the final generation. A pattern is the genotype of an individual and the corresponding rule is its phenotype, constructed using the first atom of the pattern as the head and the remaining atoms as the body.

The goal of the algorithm is to discover rules capable of making (possibly a large number of) accurate predictions. In this algorithm we use two fitness functions and each is run on an independent experiment, in which either one contains a metric suited to the OWA (Open World Assumption).

5.3.1 Representation

An evolutionary algorithm is shown by Algorithm 7, which describes the overall flow of discovering the best frequent patterns having the maximum length of a given MAX_LENGTH. The algorithm can be expressed in the following steps:

1. Firstly, initialize a list A_f of frequent atoms (line 1 in Algorithm 7), each being a pattern containing an atomic concept name or an atomic role name in the knowledge base. This list is computed once and for all before launching the evolutionary process and it maintains all atomic concept and atomic role names whose instance retrieval is higher than a threshold θ_f. This step is similar to calling the function **CreateGeneralPatterns** at the line 2 in Algorithm 1.

2. The first population *pop* with n individuals (patterns) is initialized by performing an initialization operator (see Section 5.3.2.1 for more details).

3. Lines 3-4 in Algorithm 7 are considered a support task of the selection operator (see Section 5.3.2.2 for more details). The fitness value of each pattern in the first population is computed to measure the quality of the pattern (see Section 5.3.3 for more details about the fitness function). Thereafter, we perform to sort all population individuals (patterns) by decreasing fitness

Algorithm 7: Evolutionary algorithm for the discovery of multi-relational association rules from a populated ontological KB

input : $\mathcal{K}$: ontological KB; θ_f: frequency threshold;
n: the size of the population;
p_{cross}: crossover probability; p_{ut}: mutation probability;
τ: truncation proportion; θ_{fit}: fitness threshold;

output: *pop*: set of frequent patterns discovered from $\mathcal{K}$

Initialize a list A_f of frequent atoms in $\mathcal{K}$;
Initialize a population pop of size n by using n times the function CreateNewPattern();
Compute fitness values for all of the patterns in pop;
Sort pop by decreasing fitness values;
theNumberOfGeneration $\leftarrow$ *0*;
while *theNumberOfGeneration* $<$ *MAX_GENERATIONS* **do**
  for $i = 1 \longrightarrow [\tau n]$ **do**
    CrossOver(pop[i], pop[[τn] + i]);
    CrossOver(pop[i], pop[2[τn] + i]);
    foreach *os* $\in$ *offspring* **do**
      with probability p_{mut} ***do*** *Mutate(os)*;
    end
    Compute fitness values for all of offspring;
    Add all offspring to pop;
  end
  Sort pop by decreasing fitness value;
  Remove patterns located at the end of pop so that the size of pop is exactly n;
  theNumberOfGeneration $\leftarrow$ *theNumberOfGeneration + 1*;
end
Remove inconsistent rules from the final population pop;
Remove rules where fitness value is less than θ_{fit} *from the final population pop*;
return *pop*

values, this means that we are performing to filter the best quality patterns and bring them to the top of the list.

4. We regard this evolutionary algorithm as steady-state, therefore operations are performed at each generation as follows:

   (a) We pick out the first $3\lfloor \tau n \rfloor$ individuals (patterns) in the population to apply genetic operator on them. Because the first $\lfloor \tau n \rfloor$ individuals has the best quality, therefore we apply the crossover operator to them twice in this algorithm (lines 8-9 in Algorithm 7); At first we mate each individual from 1 to $\lfloor \tau n \rfloor$ with each individual from $\lfloor \tau n + 1 \rfloor$ to $2\lfloor \tau n \rfloor$ and apply the crossover operator to them; At second we continue mating each individual from 1 to $\lfloor \tau n \rfloor$ with each individual from $\lfloor 2\lfloor \tau n \rfloor + 1 \rfloor$ to $3\lfloor \tau n \rfloor$ and apply the crossover operator to them as well (see Figure. 5.4).

   (b) After performing the crossover operator, we have two offspring (see Section 5.3.2.3 for more details), thus the total number of offspring generated per generation is $4\lfloor \tau n \rfloor$. All these offspring (patterns) are perturbed by the mutation operator (see Section 5.3.2.4 for more details) with a given probability p_{mut} (lines 10-11 in Algorithm 7) for each.

   (c) The fitness value of each offspring is computed, after that all offspring are added to the population (lines 13-14 in Algorithm 7). At this time the number of individuals in the population increases by $4\lfloor \tau n \rfloor$ individuals.

   (d) In order to keep the number of individuals in the population as the original population size and at the same time eliminate weak individuals within the population, we perform the following two processes: Sort all individuals in the population by decreasing fitness values for the former; Eliminate $4\lfloor \tau n \rfloor$ individuals located at the end of the population for the latter (lines 16-17 in Algorithm 7). With this method, we really retain the best individuals for the next generation. And obviously the best individuals in the whole process of algorithm is in the population of the final generation.

5. We achieve the best individuals by eliminating unqualified individuals in the population of the final generation. The list of the discovered frequent patterns

is the list of the rest individuals in this population. The best individual to be kept in this population must overcome two conditions as follows:

(a) The rule to be obtained from the pattern (individual) is consistent with the knowledge base (line 20 in Algorithm 7): this means that the rule is satisfiable when considered jointly with the ontology (see Section 5.3.4 for more details), $\mathcal{K} \cup p \not\models \bot$, in which p is the considered pattern in the population.

(b) The fitness value of the pattern (individual) is greater than a given threshold θ_{fit}.

5.3.2 Operators

5.3.2.1 Initialization Operator

Algorithm 8: The function CreateNewPattern()

input : a global variable A_f: a list of frequent atoms;
output: p: a new random pattern

```
1  length ← random(2, MAX_LENGTH);
2  p ← ∅;
3  while p.length() < length do
4  |   pick an atom a ∈ A_f at random;
5  |   if ¬p.isEmpty() then
6  |   |   Call AdjustAtom() in Algorithm 24 to adjust the variables in a to
   |   |     ensure the language bias is respected;
7  |   end
8  |   add a to the end of p;
9  end
10 return p
```

The initialization operator initializes the first population *pop* with n patterns by using n times the function **CreateNewPattern**.

The function **CreateNewPattern** is shown in Algorithm 8. First, this function initializes the maximum length of the pattern in range from 2 to MAX_LENGTH.

After that the function proceeds to create a new pattern by picking a frequent atom at random from the list A_f and adding that atom at the end of the pattern, this is performed iteratively until reaching the maximum length of the pattern. The picked atom can already appear in the pattern. Each time an atom is added to the pattern, the variables in this atom have to be adjusted (Section A.1) for ensuring the language bias to be respected, what means we never create a redundant or trivial rule to be obtained from the new pattern.

5.3.2.2 Selection Operator

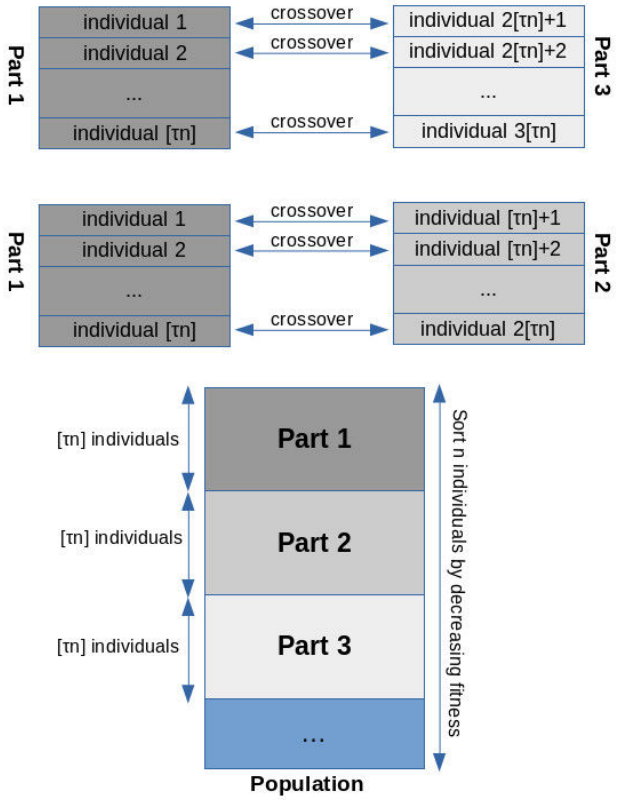


Figure 5.4: Selection operator for crossover

Before performing the selection operator, individuals in the population are sorted by decreasing fitness value and a given parameter τ ($0 < \tau < \frac{1}{3}$) is used to assist in the selection of individuals.

The selection operator is used before calling the crossover operator. The purpose of this operator is to select the best individuals in the population to perform

crossover. An illustrative diagram is depicted in Figure. 5.4: $3[\tau n]$ individuals are selected for reproduction, each individual in Part 1 (the best individuals) is selected twice to mate with each individual in Part 2 and each individual in Part 3, respectively, in order from top to bottom.

5.3.2.3 Crossover Operator

Algorithm 9: The CrossOver Operator

input : p_1, p_2: the two patterns to be crossed over.

output: O_1, O_2: two patterns that are a recombination of the input patterns.

Initialize a set L containing all of the atoms in both parents
$A_1 \leftarrow$ *a set contains atoms of* p_1
$A_2 \leftarrow$ *a set contains atoms of* p_2
$L \leftarrow A_1 \cup A_2$;
Randomly choose a target length for each offspring in the range of 2 to MAX_LENGTH
$length\_O_1 \leftarrow$ *random(2, MAX_LENGTH)*
$length\_O_2 \leftarrow$ *random(2, MAX_LENGTH)*;
$O_1 \leftarrow \emptyset$;
$O_2 \leftarrow \emptyset$;
for $i = 1 \longrightarrow 2$ **do**
    while $O_i.length() < length\_O_i$ **do**
        Pick an atom $a \in L$ *at random*;
        if $\neg O_i.isEmpty()$ **then**
            Call AdjustAtom() in Algorithm 24 to adjust the variables in a to ensure the language bias is respected;
        end
        Add a to the end of O_i;
    end
end
return O_1, O_2

The crossover operator produces two offspring patterns from two parent patterns that are selected according to the selection operator (Figure. 5.4). The atoms of the offspring are randomly selected from the atoms of the parents. The crossover operator is shown by Algorithm 9, which describes the overall flow of creating two offspring patterns having the maximum length of a given MAX_LENGTH from two parent patterns. The operator can be expressed in the following steps:

1. Initialize a set L including all the atoms in the two input patterns (parents) (line 1 in Algorithm 9). This set contains only the concept or role names of the parent patterns.

2. Initialize randomly the maximum length of the two offspring in range from 2 to MAX_LENGTH (line 2 in Algorithm 9). The maximum length can be different for each offspring.

3. The operator creates two output patterns (offspring) and each pattern is created by picking an atom at random from the set L and adding that atom to the end of the pattern, this is performed iteratively until reaching the maximum length of the pattern. The picked atom can already appear in the pattern. Each time an atom is added to the pattern, the variables in this atom have to be adjusted for ensuring the language bias to be respected.

Example 5.1. *Given the two parent patterns*

p_1: $[\mathsf{sonOf}(x, y), \mathsf{fatherOf}(y, x), \mathsf{Male}(x)]$,

p_2: $[\mathsf{fatherOf}(x, y), \mathsf{grandfatherOf}(x, z), \mathsf{sonOf}(z, y)]$,

according to Algorithm 3, we have:

$$L = \{\mathsf{sonOf}, \mathsf{fatherOf}, \mathsf{Male}, \mathsf{grandfatherOf}\}.$$

Suppose the (random) target length for the first child (O_1) is 4 and for the second child (O_2) is 3. The offspring after performing crossover may be the following patterns:

O_1: $[\mathsf{grandfatherOf}(x, y), \mathsf{fatherOf}(x, z), \mathsf{fatherOf}(z, y), \mathsf{Male}(y)]$,

O_2: $[\mathsf{fatherOf}(x, y), \mathsf{grandfatherOf}(x, z), \mathsf{fatherOf}(y, z)]$.

Algorithm 10: The Mutation Operator

input : p: the pattern to be mutated.

output: p': the mutated pattern.

```
1  if p.getFitnessValue() > θ_mut then
2      if p.length() < MAX_LENGTH then
3          p' ← Specialization(p);
4      end
5      else
6          p' ← CreateBodyPattern(p);
7      end
8  end
9  else
10     if p.length() > 2 then
11         p' ← Generalization(p);
12     end
13     else
14         p' ← CreateBodyPattern(p);
15     end
16 end
17 return p'
```

5.3.2.4 Mutation Operator

The mutation operator perturbs a pattern from the offspring of crossover with a given probability p_{mut} and uses two operators based on the idea of specialization and generalization in ILP. The mutation operator is shown by Algorithm 10, which describes the whole mutation process of the offspring being the output of crossover operator. The operator can be expressed in the following steps:

1. If the fitness value of the input pattern (offspring) is above a given threshold θ_{mut} and the length of it is smaller than MAX_LENGTH then we apply the specialization operator (see Section 5.3.2.5 for more details) on it.

2. If the fitness value of the input pattern (offspring) is below a given threshold θ_{mut} and the length of it is greater than 2 then we apply the generalization operator (see Section 5.3.2.6 for more details) on it.

3. In case a pattern is too long to undergo specialization or too short to undergo generalization, the mutation operator will apply function CreateBody-Pattern() on the input pattern, which creates a completely new body by picking atoms at random from the list A_f of frequent atoms, while keeping the same head as the the parent pattern and respecting the language bias. The picked atoms may be the same.

Example 5.2. *Assume* $p = [\mathsf{siblingOf}(x, y), \mathsf{stayWith}(y, x)]$,
with $fitness(p) < \theta_{\text{mut}}$ *undergoes mutation; then,* $p' = CreateBodyPattern(p)$,
for instance, and $p' = [\mathsf{siblingOf}(x, y), \mathsf{brotherOf}(z, x), \mathsf{Female}(x), \mathsf{sisterOf}(y, z)]$.

5.3.2.5 Specialization Operator

The specialization operator, detailed in Algorithm 11, appends a new atom by picking an atom at random from the list A_f of frequent atoms to the input pattern, while preserving the language bias.

Example 5.3 (Specialization)**.** *Given*
$p = [\mathsf{GrandFather}(x), \mathsf{Father}(x)]$,
$p' = Specialization(p)$ *might yield, for instance,*
$p' = [\mathsf{GrandFather}(x), \mathsf{Father}(x), \mathsf{Husband}(x)]$.

Algorithm 11: The Specialization Operator

input : p: the pattern to be specialized;
a global variable A_f: a list of frequent atoms;

output: p': the specialized pattern.

1 *pick an atom* $a \in A_f$ *at random*;
2 *Call AdjustAtom() in Algorithm 24 to adjust the variables in a to ensure the language bias is respected (adjust according to p)*;
3 $p' \leftarrow$ *Add a to the end of p*;
4 **return** p'

5.3.2.6 Generalization Operator

Algorithm 12: The Generalization Operator

input : p: the pattern to be generalized;

output: p': the generalized pattern.

1 *Randomly generate a number n which represents the number of atoms will be removed.*
n ← *random(1, p.body.length() - 1)*;
2 **for** $i = 1 \longrightarrow n$ **do**
3 | *Remove the last atom from p*;
4 **end**
5 $p' \leftarrow$ *Call AdjustPattern() in Algorithm 36 to adjust the variables of atoms in p (if necessary) to ensure the language bias is respected.*;
6 **return** p'

The generalization operator, detailed in Algorithm 12, removes a random number of atoms located at the end of the body of p. After removing atoms, the length of the body must remain at least one atom and preserve the language bias.

Example 5.4 (Generalization)**.** *Given*
$p = [\mathsf{sonOf}(x,y), \mathsf{fatherOf}(y,x), \mathsf{Male}(x), \mathsf{motherOf}(z,x), mathsfspouseOf(z,y)]$,
$p' = Generalization(p)$ *might yield, assuming the operator randomly chooses to remove three atoms,*
$p' = [\mathsf{sonOf}(x,y), \mathsf{fatherOf}(y,x)]$.

5.3.3 Fitness function

In this algorithm, we run on two independent experiments and the fitness function of each experiment is defined as follows:

1. The first fitness function is defined as the head coverage of the rule:

$$f_H(r) = HeadCoverage(r)$$

   This metric captures the generality of a pattern, since one expects good-quality patterns to cover a large share of the known true facts.

2. The second fitness function is defined as a combination between the head coverage and confidence of the rule:

$$f_{HPCA}(r) = HeadCoverage(r) + PCAConfidence(r)$$

   This metric allows, besides covering a large number of true facts (the head coverage metric) to also cover as few false facts as possible (the rule confidence). Since DLs adopt the OWA, we use the PCA-Confidence metric (Definition 12) to measure the confidence of a pattern. This metric is an indication of how often the pattern has been found to be true. High confidence value means that the pattern has a low error rate and vice versa. A high fitness indicates that a pattern is meaningful (general and accurate). The two terms of the fitness function might be viewed as two conflicting objectives; therefore, they could be weighted differently or a two-objective EA could be used to find non-dominated rules. We leave the exploration of both ideas for future work.

The comparison of experimental results of the above two fitness functions is shown in Section 5.4. The result of the second fitness function is better than the result of the first fitness function, which demonstrates the correctness of adding the rule PCA-Confidence to the fitness function.

5.3.4 Consistency check

For each of the obtained rules from the patterns at the final population, it is considered jointly with the ontology, if the rule is unsatisfiable then it is an inconsistent rule, otherwise it is an consistent rule.

We only check patterns for consistency in the final population, without checking them during evolution. We defer this check for the following reasons:

1. Checking rules for consistency may be computationally very expensive, to the point that the algorithm gets stuck, even with the small ontology.

2. It is not necessary to immediately omit the inconsistent rules during evolution because even if we apply a genetic operator to inconsistent rules, its offspring may still be consistent rules.

The satisfiability check in the current implementation is performed by using an off-the-shelf OWL reasoner, namely Pellet [SPG+07].

5.3.5 Language bias

In the genetic algorithm, whenever adding or removing atoms from the pattern, we must adjust the variables in order to ensure the language bias is respected, this means that the discovered patterns are not redundant and trivial. Patterns complying with the definitions in Section 3.5, are considered as respecting the language bias.

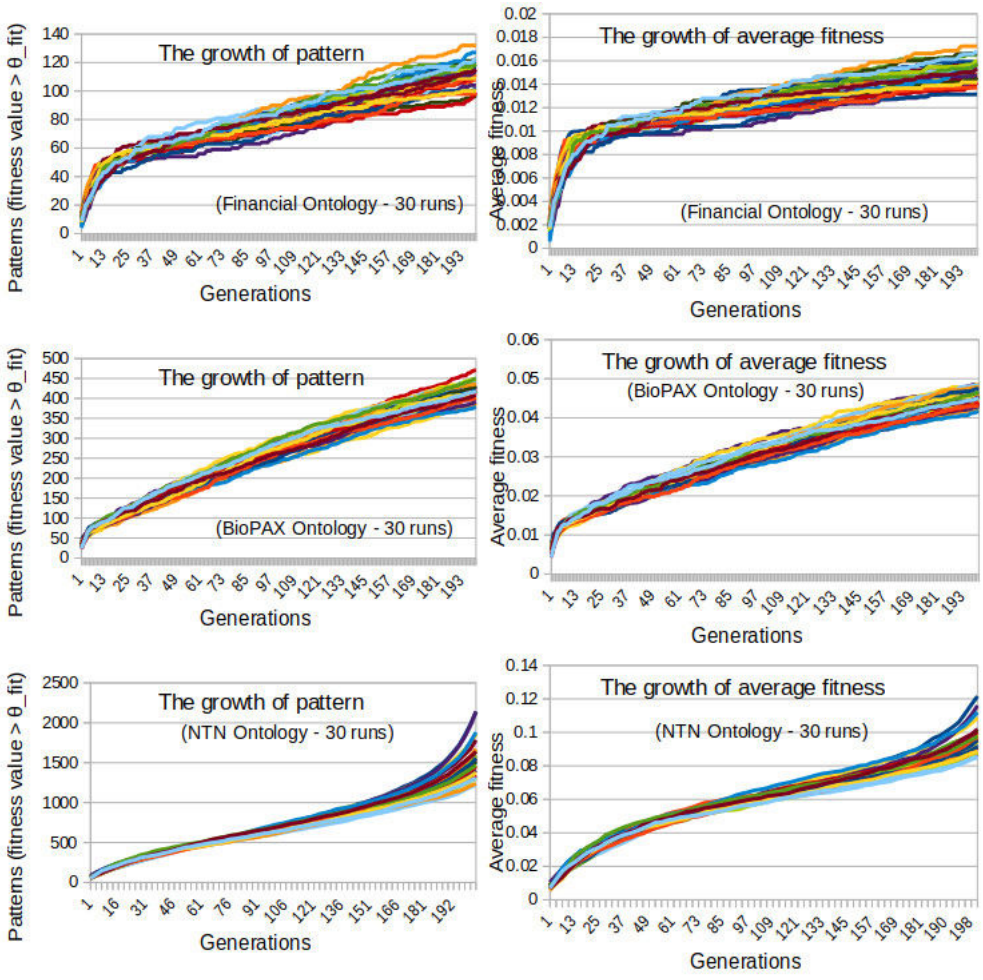
5.4 Experiments and evaluation

We carry out the experiments based on the evolutionary algorithm with respect to the publicly available ontologies shown in Table 4.1. The objectives and evaluation metrics of the experiments are shown in Section 4.4.1.

We perform two independent experiments corresponding to the two fitness functions described in Section 5.3.3. For each ontology in Table 4.1, we build three stratified samples by randomly removing respectively 20%, 30%, 40% of the concept assertions, according to Algorithm 6, thus the total number of stratified samples to be created is 9. For each experiment, we run the evolutionary algorithm presented in Section 5.3 by repeating, for each run, the sampling procedure. We performed 30 runs for each stratified sample of each ontology using the following parameter setting:

$$
\begin{array}{rclrcl}
n & = & 5,000; & p_{\text{mut}} & = & 5\%; \\
\text{MAX\_GENERATIONS} & = & 200; & \theta_{\text{mut}} & = & 0.2; \\
\text{MAX\_RULE\_LENGTH} & = & 10; & \tau & = & \frac{1}{5} \\
 & & & \theta_f & = & 1.
\end{array}
$$

Figure 5.5: The growth of population over generations



The charts in Figure. 5.5 demonstrate the unfolding of the evolutionary process over 30 distinct runs. The left charts shows the growth over generations of the number of patterns having a fitness greater than θ_{fit}. The right charts shows the growth over generations of the average fitness of the entire population. We can observe that high quality patterns are gradually discovered over the generations.

Table 5.1: Averageg (± st.dev.) performance on each ontology
($f_H(r) = HeadCoverage(r)$)

Ontology	Sample	Match Rate	Comm. Rate	Ind. Rate	Precision	Tot. nr. Predictions
Financial	20%	0.855 ± 0.033	0	0.145 ± 0.033	1.0	47,232 ± 36,777
	30%	0.864 ± 0.044	0	0.136 ± 0.044	1.0	25,456 ± 34,174
	40%	0.861 ± 0.044	0	0.139 ± 0.044	1.0	23,207 ± 30,133
BioPAX	20%	0.567 ± 0.031	0	0.433 ± 0.031	1.0	84,035 ± 15,018
	30%	0.591 ± 0.03	0	0.409 ± 0.03	1.0	85,499 ± 11,660
	40%	0.58 ± 0.027	0	0.42 ± 0.027	1.0	90,856 ± 14,048
NTNMerged	20%	0.572 ± 0.026	0	0.428 ± 0.026	1.0	2,311,624 ± 287,858
	30%	0.564 ± 0.039	0	0.436 ± 0.039	1.0	2,314,346 ± 458,522
	40%	0.621 ± 0.027	0	0.379 ± 0.027	1.0	2,345,588 ± 357,565

5.4.1 The ability to predict assertional knowledge

Results in the Tables 5.1 and 5.2 have been averaged over 30 different runs for each stratified sample. Table 5.1 offers the results of the evolutionary algorithm using the fitness function $f_H(r)$; Table 5.2 offers the results of the evolutionary algorithm using the fitness function $f_{HPCA}(r)$;

We have the following remarks when looking at the two tables (Table 5.1 and Table 5.2):

1. Positive values of *match rate* at both tables indicate that the discovered rules are actually able to predict new assertional knowledge for the considered ontologies.

2. No contradicting knowledge is predicted for both fitness functions because values of *commission error rate* are equal to 0.

3. Values of *induction rate* are greater than 0 for all ontologies. This means that

Table 5.2: Avg (± st.dev.) performance on each ontology
($f_{HPCA}(r) = HeadCoverage(r) + PCAConfidence(r)$)

Ontology	Sample	Match Rate	Comm. Rate	Ind. Rate	Precision	Total # Predictions
Financial	20%	0.871 ± 0.036	0 0	0.129 ± 0.036	1.0	44,962 ± 41,949
	30%	0.855 ± 0.047	0 0	0.145 ± 0.047	1.0	39,401 ± 44,645
	40%	0.864 ± 0.039	0 0	0.136 ± 0.039	1.0	31,226 ± 33,952
BioPAX	20%	0.571 ± 0.028	0 0	0.429 ± 0.028	1.0	86,920 ± 11,691
	30%	0.59 ± 0.025	0 0	0.41 ± 0.025	1.0	79,543 ± 11,850
	40%	0.584 ± 0.031	0 0	0.416 ± 0.031	1.0	97,559 ± 13,049
NTNMerged	20%	0.632 ± 0.059	0 0	0.368 ± 0.059	1.0	3,439,660 ± 554,720
	30%	0.6 ± 0.055	0 0	0.4 ± 0.055	1.0	2,353,420 ± 477,735
	40%	0.711 ± 0.075	0 0	0.289 ± 0.075	1.0	2,899,464 ± 563,711

evolutionary algorithm with the above fitness functions can come up with rules that induce previously unknown facts.

4. Values of precision are always equal to 1 at both tables on all samples of all ontologies considered, which show that these results fully confirm the capability of the proposed approach to discover accurate rules.

5. We notice that the evolutionary algorithm approach creates a very large number of predictions compared to the level-wise generate-and-test approach (Table 4.2).

5.4.2 The importance and the value added

We call GA_F_{HPCA} the evolutionary algorithm with the fitness function $f_H(r)$; GA_F_H the evolutionary algorithm with the fitness function $f_{HPCA}(r)$; LW-GAT the level-wise generate-and-test method presenting in Chapter 4; AMIE [GTHS13]

Table 5.3: Comparison of the number of discovered rules.

Ontology	Samp.	# The total number of rules discovered			
		GA_F_{HPCA}	GA_F_H	LW-GAT	AMIE
Financial	20%	27 ± 3	26 ± 4	**177**	2
	30%	26 ± 3	25 ± 3	**181**	2
	40%	24 ± 4	23 ± 3	**180**	2
BioPAX	20%	132 ± 10	129 ± 13	**298**	8
	30%	118 ± 12	128 ± 9	**283**	8
	40%	137 ± 12	129 ± 11	**272**	0
NTNMerged	20%	**1,834** ± 782	1,157 ± 168	243	1,129
	30%	**1,235** ± 495	1,052 ± 353	225	1,022
	40%	**1,810** ± 733	1,088 ± 181	239	1,063

the state-of-the-art system. Table 5.3 reports the average number of rules discovered by each system given each knowledge base sample. It is possible to see that LW-GAT outperformed the number of rules discovered by GA_F_{HPCA}, GA_F_H and AMIE for the case of Financial and BioPAX ontologies. The reason is that the output rules (Definition ...) of LW-GAT is open rules, while the output rules of the rest are close rules (Definition ...). We can observe a clear tendency for GA_F_{HPCA} to perform better with respect to the ontology containing large number of assertions; GA_F_H performs better than LW-GAT and AMIE. A dominant feature of GA_F_{HPCA} and GA_F_H is that the maximum length of a discovered rule is 10, while LW-GAT and AMIE are only 3.

Additional comparative results are reported in Table 5.4. Here, given the top m rules, with n equal to 10 or equal to the number of rules discovered by AMIE, when

Table 5.4: Comparison of the number of extracted predictions.

Ontology	Samp.	Top				
		n	# Predictions			
			GA_F_{HPCA}	GA_F_H	LW-GAT	AMIE
Financial	20%	2	**42,575 *** ± 38,239	46,536 ± 36,708	29	208
	30%	2	**36,799** ± 41,667	23,764 ± 32,093	57	197
	40%	2	**30,263** ± 33,825	22,316 ± 30,120	85	184
BioPAX	20%	8	**41,024 *** ± 7,567	40,051 ± 8,213	25	2
	30%	8	**39,283 *** ± 6,485	40,580 ± 6,562	34	2
	40%	8	**43,698 *** ± 6,524	40,860 ± 8,052	50	0
NTNMerged	20%	10	**933,248 *** ± 110,786	593,634 ± 81,448	620	420
	30%	10	**724,020 *** ± 162,851	646,419 ± 99,601	623	281
	40%	10	**828,317 *** ± 250,804	707,093 ± 154,251	625	332

fewer than 10 rules were discovered, the number of correct predictions generated by the top m rules discovered by each system is compared. The results reported in Table 5.4 corroborate the claim that GA_F_{HPCA} can substantially outperform the existing systems, not only in terms of rules discovered, but also (and more importantly) in terms of their predictive power. A star in the GA_F_{HPCA} column means that Welch's t-test on the comparison of GA_F_{HPCA} and GA_F_H rejects the null hypothesis with a confidence level of at least 95%.

5.5 Summary

In this chapter, we presented an evolutionary algorithm for discovering multi-relational association rules from ontological knowledge bases, to be used for enriching assertional knowledge. This algorithm is an improvement over the level-wise generated-and-test algorithm. The algorithm maintains a population of the patterns and makes it evolve by iteratively performing genetic operators and discovered rules are gained at the final population after checking consistency with the ontology.

Discovered rules are coded in SWRL, hence they can be directly added to the considered ontological KB deriving new assertional knowledge. Furthermore, the discovered rules may suggest new axioms at schema level, such as role transitivity, symmetry, role/concept subsumption.

We applied the proposed approach in two independent experiments, each with a separated fitness function. This experiments has been evaluated through its application to publicly available ontologies and comparisons with the state of the art system.

Chapter 6

Comparing Rule Evaluation Metrics

Contents

6.1 Introduction 87

6.2 Experiment and evaluation 88

6.2.1 Compare the number of rules discovered and the ability to predict assertional knowledge 89

6.2.2 The importance and the value added 91

6.3 Summary 92

6.1 Introduction

In Chapter 5, we performed and compared two independent experiments of the evolutionary algorithm with two fitness functions representing two metrics. In this chapter, we will study the predictive ability of some other metrics, originally proposed for scoring association rules. As in Chapter 5, we will employ these metrics as building blocks for a fitness function for evolutionary inductive programming.

All metrics to be used in experiments of this chapter are popular asymmetric metrics which are shown in Sections 3.6 and 3.7. We employ asymmetric metrics because our evolutionary algorithm searches for hidden knowledge patterns in the form of SWRL rules having the general form A $\longrightarrow$ B and values of an asymmetric metric for A $\longrightarrow$ B and B $\longrightarrow$ A may not be the same.

The main goal in this chapter is that we might select metrics that are suitable with data semantics by comparing the number of generated rules, total predictions and the number of unknown facts when the metrics are used to compute the fitness

function of the evolutionary algorithm. The selection of these metrics is also a crucial step towards scoring multi-relational association rules that are generated from ontologies.

Because the evolutionary algorithm is presented in Section 5.2 along with the asymmetric metrics are mentioned in Section 3.6, this chapter is shortened as follow: Experiments and evaluation are given in Section 6.2. Conclusions are drawn in Section 6.3.

6.2 Experiment and evaluation

In order to easily compare results with the experiments in previous chapters, we continue to carry out experiments on the ontologies publicly available in Table 4.1. The objectives and evaluation metrics of the experiments are shown in the Section 4.4.1.

We perform independent experiments for separate fitness functions, each fitness function corresponds to an asymmetric metric describing in Sections 3.6 and 3.7. We construct 3 stratified samples of each ontology in Table 4.1 by randomly removing respectively 20%, 30%, 40% of the concept assertions, according to Algorithm 6. We performed 30 runs of the evolutionary algorithm described in Section 5.2 for each stratified sample and for each choice of fitness function using the following parameter setting:

$$
\begin{aligned}
n &= 5,000; & p_{\text{mut}} &= 5\%;\\
\text{MAX\_GENERATIONS} &= 200; & \theta_{\text{mut}} &= 0.2;\\
\text{MAX\_RULE\_LENGTH} &= 10; & \tau &= \tfrac{1}{5}\\
\theta_{\text{fit}} &= 0 & \theta_f &= 1.\\
(\text{Conviction: } \theta_{\text{fit}} &= 1)
\end{aligned}
$$

As we have known, θ_{fit} is a given threshold used to aid in pattern selection. For each fitness function, θ_{fit} might be set to a new value within the effective range that corresponds to the metric used to compute the fitness function. Details of the effective range are presented in Table 6.1.

Table 6.1: Symbols and range of metrics (the effective range is used to assist in the choice of θ_{fit}.)

Symbol	Metric	Range	Effective range
H	Head Coverage	[0, 1]	(0, 1]
C	Confidence	[0, 1]	(0, 1]
P	PCA-Confidence	[0, 1]	(0, 1]
L	Laplace	[0, 1]	(0, 1]
CV	Conviction	[0.5, $+\infty$)	(1, $+\infty$)
CF	Certainty Factor	[-1, 1]	(0, 1]
A	Added Value	[-0.5 , 1]	(0, 1]
J	J-Measure	[0, 1]	(0, 1]
G	Gini Index	[0, 1]	(0, 1]

6.2.1 Compare the number of rules discovered and the ability to predict assertional knowledge

Our experiments aimed at comparing the results obtained by the EA using different rule evaluation metrics as fitness based on the three following criteria:

1. The *number of the rules discovered* by the EA.
2. The *induction rate*: if > 0, this means assertions are predicted that could not be inferred from the stratified version. The higher the induction rate, the more novel predictions (unknown facts) are induced for the KB.
3. The *number of correct predictions* = *number of predictions* $\times$ *precision*, where the *number of predictions* is the number of predicted assertions and *precision* is defined in Definition 19.

Table 6.2 shows a comparison of the metrics (identified with the acronyms defined in Table 6.1) according to the first criterion. The second and third criteria are used to compare the predictive power of the discovered rules. In order to compare the metrics according to these criteria, we applied the rules discovered from the stratified samples to the full ontology versions and collected all predictions, i.e., the head atoms of the instantiated rules. Given the collected predictions, those already contained in the stratified ontology samples were discarded, while the remaining predicted facts were considered. A prediction is evaluated as *correct* if it is contained/entailed by the full ontology version and as *incorrect* if it is inconsistent with

Table 6.2: Comparison of the metrics by the number of discovered rules.

Ontology	Samp.	Total number of discovered rules by metric ± stdev								
		H	C	P	L	CV	CF	A	J	G
Financial	20%	26 ± 4	25 ± 4	25 ± 3	3,254 ± 30	4 ± 1	25 ± 3	26 ± 3	3 ± 1	487 ± 12
	30%	25 ± 3	25 ± 4	25 ± 4	3,301 ± 31	4 ± 1	26 ± 3	24 ± 4	4 ± 1	485 ± 9
	40%	23 ± 3	23 ± 3	22 ± 4	3,296 ± 31	3 ± 1	23 ± 4	21 ± 3	3 ± 1	479 ± 11
Biopax	20%	129 ± 13	122 ± 12	130 ± 10	4,293 ± 24	35 ± 5	118 ± 9	119 ± 9	58 ± 5	3,486 ± 182
	30%	128 ± 9	130 ± 13	130 ± 9	4,384 ± 22	33 ± 5	117 ± 8	110 ± 9	55 ± 5	3,658 ± 139
	40%	129 ± 11	136 ± 11	133 ± 8	4,530 ± 23	36 ± 5	124 ± 9	122 ± 7	59 ± 6	3,560 ± 157
NTNMerged	20%	1,157 ± 168	1,345 ± 423	1,418 ± 492	4,563 ± 53	382 ± 31	671 ± 36	656 ± 34	504 ± 22	2,040 ± 690
	30%	1,052 ± 353	947 ± 238	1,017 ± 370	4,805 ± 13	509 ± 39	743 ± 45	728 ± 48	460 ± 21	457 ± 90
	40%	1,088 ± 181	1,223 ± 177	1,295 ± 357	4,797 ± 22	397 ± 26	687 ± 38	664 ± 34	500 ± 26	1,506 ± 61

the full ontology version. All the results (see Table 6.3, 6.4, 6.5, 6.6 and 6.7) have been computed using the rules discovered by each metrics (see Table 6.2) based on 30 runs with the above parameter setting and have been measured in terms of *precision* (see Definition 19), *match*, *commission*, and *induction rate* (see Section 3.8). The statistic significance of all pairwise comparisons between metrics have been assessed using 1-tailed Welch's t-test.

The evolutionary algorithm achieves *precision* = 1 and *commission error rate* = 0 on all versions of all considered ontologies; this confirms its ability to discover accurate rules; as a consequence, the *number of correct predictions* coincides with the number of discovered predictions.

From the observations in Tables 6.2, 6.3, 6.4, 6.5, 6.6 and 6.7, we can draw a few remarks:

1. *Laplace* has the highest number of discovered rules. However, it hardly produces any new knowledge (induction rate ≈ 0).

2. *Gini Index* scores the second highest number of discovered rules. However,

this measure looks less robust when compared to other metrics, since large deviations among discovered rules show up for different stratified samples of the same ontology (see Table 6.2); sometimes, it produces much new knowledge, sometimes little or none (induction rate is not stable - see Table 6.7). In addition, the number of predictions is medium or low compared to other metrics (see Table 6.7).

3. Five metrics (*HeadCoverage*, *Confidence*, *PCA-Confidence*, *Certainty Factor*, and *Added Value*) allow the evolutionary algorithm to discover the largest number of rules (see Table 6.2) and, which is even more relevant, to come up with rules that induce a large number of previously unknown facts (induction rate > 0), with a very large absolute number of correct predictions (see Tables 6.3, 6.4, 6.5 and 6.6).

4. Two metrics (*Conviction* and *J-Measure*) produce the smallest number of rules. Although both the induction rate and the number of predictions are acceptable, the low number of discovered rules may mean valuable rules are missed out.

From the above remarks, we may conclude that *HeadCoverage*, *Confidence*, *PCA-Confidence*, *Certainty Factor*, and *Added Value* are the best choices as an optimization criterion (i.e., fitness function).

6.2.2 The importance and the value added

We call GA_F_{HPCA} the evolutionary algorithm with the fitness function $f_{HPCA}(r) = HeadCoverage(r) + PCAConfidence(r)$ presenting in Chapter 5; LW-GAT the level-wise generate-and-test method presenting in Chapter 4; AMIE [GTHS13] the state-of-the-art system. We compared the experimental performance of the five best metrics to GA_F_{HPCA}, LW-GAT and AMIE which are closest to it in purpose. Table 6.8 reports the number of rules discovered by each system given each knowledge base sample. We can remark the following:

1. The top-5 metrics discover more rules than LW-GAT from NTNMerged, but fewer from the Financial and Biopax knowledge bases. One reason is that LW-GAT can discover also *open* rules (see Example 3.9), which are barred

by language bias of the evolutionary algorithm; furthermore, the maximum length of a rule is 10 atoms. Another reason is that the number of individuals in Financial and Biopax is less than that of NTNMerged (see Tab 4.1, last column). If one factor these differences out, the top-5 metrics are superior to LW-GAT.

2. The top-5 metrics discover more rules than AMIE from Financial and Biopax and a comparable number from the NTNMerged knowledge base. One limitation of deterministic level-wise generate-and-test methods like AMIE and LW-GAT is that they cannot scale up to rules longer than 3 atoms, while the evolutionary algorithm (with any metrics) can easily discover rules of 10 atoms (and possibly more).

3. The fitness function of GA_F_{HPCA} outperforms each of the top-5 metrics; however, it is a combination of two of them. This suggests a new promising direction of research, that is to try to find an optimal fitness function for the evolutionary algorithm by combining the individual metrics studies in this chapter.

6.3 Summary

We have just compared the results of the experiments in the previous chapters with some popular asymmetric metrics by applying them as fitness functions for evolutionary inductive programming. After comparison, we figured out five metrics as the most promising candidates for further exploration.

Through this comparison, we also identified a promising new direction for research with the aim of optimizing the fitness function. That is to find ways to combine the above metrics to achieve the best results.

Table 6.3: Avg (± st.dev.) performance on each ontology of HeadCoverage (H), Confidence (C)

	Ontology	Samp.	Match Rate	Com. Rate	Ind. Rate	Precision	Total # Predictions
H	Financial	20%	0.855 ± 0.033	0	0.145 ± 0.033	1.0	47,232 ± 36,777
		30%	0.864 ± 0.044	0	0.136 ± 0.044	1.0	25,456 ± 34,174
		40%	0.861 ± 0.044	0	0.139 ± 0.044	1.0	23,207 ± 30,133
	BioPAX	20%	0.567 ± 0.031	0	0.433 ± 0.031	1.0	84,035 ± 15,018
		30%	0.591 ± 0.03	0	0.409 ± 0.03	1.0	85,499 ± 11,660
		40%	0.58 ± 0.027	0	0.42 ± 0.027	1.0	90,856 ± 14,048
	NTNMerged	20%	0.572 ± 0.026	0	0.428 ± 0.026	1.0	2,311,624 ± 287,858
		30%	0.564 ± 0.039	0	0.436 ± 0.039	1.0	2,314,346 ± 458,522
		40%	0.621 ± 0.027	0	0.379 ± 0.027	1.0	2,345,588 ± 357,565
C	Financial	20%	0.848 ± 0.045	0	0.152 ± 0.045	1.0	43,151 ± 44,254
		30%	0.860 ± 0.038	0	0.140 ± 0.038	1.0	27,589 ± 41,184
		40%	0.858 ± 0.051	0	0.142 ± 0.051	1.0	33,795 ± 41,880
	BioPAX	20%	0.574 ± 0.036	0	0.426 ± 0.036	1.0	79,454 ± 14,019
		30%	0.584 ± 0.027	0	0.416 ± 0.027	1.0	88,879 ± 12,890
		40%	0.582 ± 0.023	0	0.418 ± 0.023	1.0	96,884 ± 13,782
	NTNMerged	20%	0.618 ± 0.042	0	0.382 ± 0.042	1.0	1,437,868 ± 253,206
		30%	0.581 ± 0.036	0	0.419 ± 0.036	1.0	1,164,306 ± 167,173
		40%	0.670 ± 0.030	0	0.330 ± 0.030	1.0	1,557,516 ± 280,666

Table 6.4: Avg (± st.dev.) performance on each ontology of PCA-Confidence (P), Laplace(L)

	Ontology	Samp.	Match Rate	Com. Rate	Ind. Rate	Precision	Total # Predictions
P	Financial	20%	0.859 ± 0.055	0	0.141 ± 0.055	1.0	41,350 ± 46,196
		30%	0.850 ± 0.055	0	0.150 ± 0.055	1.0	32,812 ± 41,501
		40%	0.859 ± 0.043	0	0.141 ± 0.043	1.0	29,762 ± 35,582
	BioPAX	20%	0.571 ± 0.028	0	0.429 ± 0.028	1.0	89,486 ± 11,303
		30%	0.584 ± 0.023	0	0.416 ± 0.023	1.0	92,392 ± 13,878
		40%	0.587 ± 0.027	0	0.413 ± 0.027	1.0	91,849 ± 11,960
	NTNMerged	20%	0.609 ± 0.046	0	0.391 ± 0.046	1.0	2,130,947 ± 380,546
		30%	0.588 ± 0.043	0	0.412 ± 0.043	1.0	1,409,235 ± 286,439
		40%	0.670 ± 0.042	0	0.330 ± 0.042	1.0	1,727,343 ± 262,891
L	Financial	20%	1.0	0	0	1.0	122,432 ± 1,704
		30%	1.0	0	0	1.0	180,231 ± 2,801
		40%	1.0	0	0	1.0	230,736 ± 3,484
	BioPAX	20%	1.0	0	0	1.0	51,060 ± 866
		30%	1.0	0	0	1.0	78,488 ± 1,527
		40%	1.0	0	0	1.0	100,699 ± 1,600
	NTNMerged	20%	0.994 ± 0.001	0	0.006 ± 0.001	1.0	197,374 ± 6,116
		30%	0.995	0	0.005	1.0	284,065 ± 5,806
		40%	0.996	0	0.004	1.0	323,085 ± 6,359

Table 6.5: Avg (± st.dev.) performance on each ontology of Conviction(CV), Certainty Factor(CF)

	Ontology	Samp.	Match Rate	Com. Rate	Ind. Rate	Precision	Total # Predictions
CV	Financial	20%	0 ± 0.001	0	1.0 ± 0.001	1.0	48,661 ± 41,318
		30%	0.001 ± 0.001	0	0.999 ± 0.001	1.0	43,078 ± 39,328
		40%	0.001 ± 0.002	0	0.999 ± 0.002	1.0	26,268 ± 33,679
	BioPAX	20%	0.08 ± 0.018	0	0.92 ± 0.018	1.0	44,971 ± 10,928
		30%	0.11 ± 0.017	0	0.89 ± 0.017	1.0	44,451 ± 10,557
		40%	0.102 ± 0.018	0	0.898 ± 0.018	1.0	50,457 ± 12,368
	NTNMerged	20%	0.32 ± 0.019	0	0.68 ± 0.019	1.0	831,416 ± 183,095
		30%	0.344 ± 0.013	0	0.656 ± 0.013	1.0	1,123,266 ± 208,471
		40%	0.361 ± 0.015	0	0.639 ± 0.015	1.0	868,467 ± 174,865
CF	Financial	20%	0.877 ± 0.038	0	0.123 ± 0.038	1.0	31,656 ± 45,045
		30%	0.852 ± 0.057	0	0.148 ± 0.057	1.0	48,568 ± 45,051
		40%	0.857 ± 0.039	0	0.143 ± 0.039	1.0	31,068 ± 36,044
	BioPAX	20%	0.556 ± 0.026	0	0.444 ± 0.026	1.0	80,361 ± 8,700
		30%	0.581 ± 0.023	0	0.419 ± 0.023	1.0	78,933 ± 11,147
		40%	0.564 ± 0.035	0	0.436 ± 0.035	1.0	84,476 ± 12,647
	NTNMerged	20%	0.565 ± 0.01	0	0.435 ± 0.01	1.0	1,039,112 ± 179,322
		30%	0.535 ± 0.014	0	0.465 ± 0.014	1.0	1,424,334 ± 180,205
		40%	0.557 ± 0.018	0	0.443 ± 0.018	1.0	2,110,928 ± 423,539

Table 6.6: Avg (± st.dev.) performance on each ontology of Added value (A), J-Measure (J)

	Ontology	Samp.	Match Rate	Com. Rate	Ind. Rate	Precision	Total # Predictions
A	Financial	20%	0.859 ± 0.041	0	0.141 ± 0.041	1.0	33,358 ± 31,445
		30%	0.858 ± 0.041	0	0.142 ± 0.041	1.0	29,866 ± 31,123
		40%	0.859 ± 0.041	0	0.141 ± 0.041	1.0	29,870 ± 44,276
	BioPAX	20%	0.549 ± 0.032	0	0.451 ± 0.032	1.0	83,666 ± 11,663
		30%	0.578 ± 0.029	0	0.422 ± 0.029	1.0	78,059 ± 9,368
		40%	0.579 ± 0.02	0	0.421 ± 0.02	1.0	84,483 ± 10,376
	NTNMerged	20%	0.563 ± 0.012	0	0.437 ± 0.012	1.0	966,840 ± 204,430
		30%	0.541 ± 0.014	0	0.459 ± 0.014	1.0	1,324,518 ± 282,410
		40%	0.566 ± 0.014	0	0.434 ± 0.014	1.0	1,632,633 ± 218,033
J	Financial	20%	0 ± 0.001	0	1.0 ± 0.001	1.0	10,148 ± 13,149
		30%	0.001 ± 0.001	0	0.999 ± 0.001	1.0	32,052 ± 39,154
		40%	0	0	1.0	1.0	36,204 ± 40,910
	BioPAX	20%	0.083 ± 0.011	0	0.917 ± 0.011	1.0	82,799 ± 11,596
		30%	0.108 ± 0.013	0	0.892 ± 0.013	1.0	80,797 ± 13,564
		40%	0.11 ± 0.013	0	0.89 ± 0.013	1.0	90,480 ± 12,579
	NTNMerged	20%	0.294 ± 0.008	0	0.706 ± 0.008	1.0	1,317,526 ± 207,005
		30%	0.301 ± 0.011	0	0.699 ± 0.011	1.0	1,765,003 ± 242,269
		40%	0.319 ± 0.011	0	0.681 ± 0.011	1.0	2,387,450 ± 698,911

Table 6.7: Avg (± st.dev.) performance on each ontology of Gini factor (G)

	Ontology	Samp.	Match Rate	Com. Rate	Ind. Rate	Precision	Total # Predictions
G	Financial	20%	0.182 ± 0.007	0	0.818 ± 0.007	1.0	20,321 ± 22,967
		30%	0.181 ± 0.01	0	0.819 ± 0.01	1.0	49,443 ± 42,556
		40%	0.186 ± 0.009	0	0.814 ± 0.009	1.0	20,645 ± 18,367
	BioPAX	20%	1.0	0	0	1.0	30,839 ± 1,632
		30%	1.0	0	0	1.0	45,063 ± 1,727
		40%	1.0	0	0	1.0	62,941 ± 2,781
	NTNMerged	20%	0.768 ± 0.054	0	0.232 ± 0.054	1.0	199,745 ± 50,410
		30%	0.725 ± 0.023	0	0.275 ± 0.023	1.0	82,059 ± 13,066
		40%	0.785 ± 0.007	0	0.215 ± 0.007	1.0	258,454 ± 10,162

Table 6.8: Comparison of the number of discovered rules.

Ontology	Samp.	# The total number of rules discovered							
		H	C	P	CF	A	GA_F_{HPCA}	LW-GAT	AMIE
Financial	20%	26 ± 4	25 ± 4	25 ± 3	25 ± 3	26 ± 3	27 ± 3	177	2
	30%	25 ± 3	25 ± 4	25 ± 4	26 ± 3	24 ± 4	26 ± 3	181	2
	40%	23 ± 3	23 ± 3	22 ± 4	23 ± 3	21 ± 3	24 ± 4	180	2
Biopax	20%	129 ± 13	122 ± 12	130 ± 10	118 ± 9	119 ± 9	132 ± 10	298	8
	30%	128 ± 9	130 ± 13	130 ± 9	117 ± 8	110 ± 9	118 ± 12	283	8
	40%	129 ± 11	136 ± 11	133 ± 8	124 ± 9	122 ± 7	137 ± 12	272	0
NTNMerged	20%	1,157 ± 168	1,345 ± 432	1,418 ± 492	671 ± 36	656 ± 34	1,834 ± 782	243	1,129
	30%	1,052 ± 353	947 ± 238	1,017 ± 370	743 ± 45	728 ± 48	1,235 ± 495	225	1,022
	40%	1,088 ± 181	1,223 ± 177	1,295 ± 357	687 ± 38	664 ± 34	1,810 ± 733	239	1,063

Chapter 7

Query Optimization

Contents

7.1 Introduction 99
7.2 From SWRL rule to SPARQL-DL Query 101
7.2.1 SPARQL-DL Query 101
7.2.2 Calculate metrics with SPARQL-DL 101
7.3 The algorithm 103
7.3.1 Representation 104
7.3.2 The individual matrix "Add a concept atom" 106
7.3.3 The individual matrix "Add a role atom" 108
7.4 How to compute metrics with the query optimization algorithm 119
7.4.1 Support 120
7.4.2 Head Coverage 121
7.4.3 Confidence 122
7.4.4 PCA-Confidence 123
7.5 Evaluation 123

7.1 Introduction

Since rules are represented in SWRL, to compute metrics for a rule, we can directly add it to the ontology and then implement a query along with the reasoner in order to retrieve calculation support information accordingly. Initially, we performed querying on SPARQL-DL (see Section 7.2.1) based on the following advantages:

1. SPARQL-DL is quite more expressive than other DL query languages (particularly, it allows to mix TBox, RBox, and ABox queries) and can still be

implemented without too much effort on top of existing OWL-DL reasoners [SP07].

2. It is easy to convert the format of a SWRL rule into the SPARQL-DL query language to gain the necessary information related to the rule.

However, creating and evaluating a large number of rules in a short time interval requires fast data retrieval speed, but the data query speed of SPARQL-DL is quite slow, although we tried to employ some methods of tuning SPARQL-DL queries for performance. Thus the performance problem is a major drawback of SPARQL-DL in evaluating the rules and the use of SPARQL-DL queries to evaluate rules is not appropriate in this case.

In order to overcome this obstacle, we propose to build an algorithm to compute metrics with the goal of improving the speed of evaluating the rules (see Section 7.3 for more details). This algorithm is derived from the hash join algorithm used in the implementation of RDBMS (Relational Database Management System) because there are some steps in the algorithm that are similar to compute the natural join of two relations. Specifically, we performed this by loading the data of the ontology into memory for processing. The in-memory data include concept names, role names and assertions to be used as the input of the algorithm. The algorithm does not load the T-Box axioms of the ontology into memory because it will invoke the reasoner to do reasoning as needed.

Performance of rule evaluation is greatly improved by this approach (for the evaluation in Section 7.5), which leads to overall system performance increasing efficiency. The limitation of the algorithm is that we need a large enough memory to store and support the calculation, thus this approach is suitable for small to medium ontologies.

The rest of this chapter is structured as follows: In Section 7.2, we introduce the SPARQL-DL query language and how the metrics are calculated based on it. An algorithm instead of querying via SPARQL-DL to improve the speed of assessing the rules is shown in Section 7.3. The application for the calculation of metrics based on the query optimization algorithm is presented in Section 7.4. Finally, evaluation and comparison are illustrated in Section 7.5.

7.2 From SWRL rule to SPARQL-DL Query

7.2.1 SPARQL-DL Query

Figure 7.1: Selection operator for crossover

As remarked in [SP07], the SPARQL-DL query language is a distinct subset of SPARQL (a query language for RDF), and is located on the top of the OWL API (a high level Application Programming Interface for working with OWL ontologies).

SPARQL-DL is defined as a powerful and expressive query language for OWL-DL that can blend TBox, RBox and ABox queries. In addition, it can interact with applications on the Semantic Web (see Figure. 7.1) and especially it can be easily built on top of existing OWL-DL reasoners.

Example 7.1. *Here, we give two examples that show SPARQL-DL is friendly and easy to use.*

1. *Query that gets all concept names:*
   SELECT ?c WHERE { Class(?c) }

2. *Query that gets the class hierarchy by means of class/sub-class pairs:*
   SELECT ?a ?b
   WHERE SubClassOf(?a, ?b)

7.2.2 Calculate metrics with SPARQL-DL

We notice that all metrics shown in Sections 3.6 and 3.7 are calculated based on a few basic components, thus we can compute those by SPARQL-DL queries and then combine their resulting values together by using the quantity of elements to be queried in order to retrieve the value of the metrics. Specifically, suppose given a rule $r = B_1 \wedge \ldots \wedge B_n \rightarrow H$, we use SPARQL-DL queries to calculate the following

four basic components in advance: $\Sigma_H(r)$, $E_H(r)$, $M_H(r)$ and $P_H(r)$, and then compute the metrics by using the number of elements of these components:

* $\text{supp}(r) = |E_H(r)|$
* $\text{headCoverage}(r) = |E_H(r)|/|\Sigma_H(r)|$
* $\text{conf}(r) = |E_H(r)|/|M_H(r)|$
* etc ...

Example 7.2. *Given a SWRL rule of the form:*

r: isMotherInLaw(x, y) $\leftarrow$ ***isHusband(y,z)*** $\wedge$ ***isMother(x,z)*** $\wedge$ ***Female(z)***

where

* ***isMotherInLaw(x, y)*** *is the rule head*
* ***isHusband(y, z)*** $\wedge$ ***isMother(x, z)*** $\wedge$ ***Female(z)*** *is the rule body*
* ***isMotherInLaw***, ***isHusband***, ***isMother*** *and* ***Female*** *are atoms*
* ***x***, ***y***, ***z*** *are variables.*

We use SPARQL-DL queries to compute the four basic components with respect to the rule ***r*** *as follows:*

1. $\Sigma_H(r)$ =
   PREFIX ex: <http://example.com#>
   SELECT ?x ?y
   WHERE { PropertyValue(?x, ex:isMotherInLaw, ?y) }
2. $E_H(r)$ =
   PREFIX ex: <http://example.com#>
   SELECT DISTINCT ?x ?y
   WHERE { PropertyValue(?x, ex:isMotherInLaw, ?y),
   PropertyValue(?y, ex:isHusband, ?z),
   PropertyValue(?x, ex:isMother, ?z),
   Type(?z, ex:Female) }

3. $M_H(r)$ =
PREFIX ex: <http://example.com#>
SELECT DISTINCT ?x ?y
WHERE { PropertyValue(?y, ex:isHusband, ?z),
PropertyValue(?x, ex:isMother, ?z),
Type(?z, ex:Female) }

4. $P_H(r)$ =
PREFIX ex: <http://example.com#>
SELECT DISTINCT ?x ?y
WHERE { PropertyValue(?y, ex:isHusband, ?z),
PropertyValue(?x, ex:isMother, ?z),
Type(?z, ex:Female),
PropertyValue(?x, ex:isMotherInLaw, ?y') }

Example 7.2 shows that it is easy to perform the calculation of metrics using SPARQL-DL query. However, the performance problem is not solved in this case due to the fact that data query speed of SPARQL-DL is quite slow. Therefore, we propose an algorithm in Section 7.3 to compute metrics in order to replace the SPARQL-DL query for the purpose speeding up the rule evaluation process. Actually, SPARQL-DL query is rewritten into an equivalent algorithm with immutable semantics.

7.3 The algorithm

The goal of this algorithm is to support the calculation of metrics instead of SPARQL-DL query to improve the system performance. The following definitions are used in the algorithm.

Definition 23 (Variable structure)**.** *A variable structure includes two components: the variable name and the list of individuals of that variable.*

Example 7.3. *One variable structure contains a variable* ***?y*** *and its list of individuals in order* ***[b, c, d]***

$?y$
b
c
d

Definition 24 (Individual Matrix)**.** *A list of variable structures is called an individual matrix.*

In fact, an individual matrix is a set of distinct bindings of the variables occurring in both the head and the body of the rule. A row in this matrix is a variable binding (Section 3.6).

Example 7.4. *The individual matrix* $Matrix_{input}$ *contains a list of variable structures in order* ***[?x, ?y,*** $?z_1$**,** $?z_2$***]*** .

$Matrix_{input}$			
$?x$	$?y$	$?z_1$	$?z_2$
a	b	a	b
k	f	e	c
c	b	b	g
s	a	s	b

The input of the algorithm is a pattern representing a SWRL rule, where the first atom is the rule head and the remaining atoms are the rule body. The output of the algorithm is an individual matrix containing a list of variable structures whose variable names are in the atoms of the input pattern. This output individual matrix has a few features as follows:

1. Used to aid in the calculation of metrics.
2. The number of individuals of all variable structures are always the same.
3. Variable names at the top of the list of variable structures are always variable names of the head of the rule to be obtained from the input pattern.

7.3.1 Representation

Algorithm 13 describes the main idea of the problem, which is to create an individual matrix from an input pattern in order to support the calculation of metrics. We

Algorithm 13: Compute an individual matrix of the pattern

input : p: the pattern to be computed.

output: $matrix$: a individual matrix

```
1  matrix ← Ø;
2  for i = 1 ⟶ p.size() do
3      if p[i].isConcept() then
4          var_struct ← a variable structure with the variable name to be the
             variable name of p[i] and the list of individuals to be the list of
             individuals of the atom p[i] obtained from the OWL reasoner;
5          matrix ← AddConceptToIndividualMatrix(matrix, var_struct);
6      end
7      else if p[i].isRole() then
8          varDomain_struct ← a variable structure with the variable name to be
             the domain variable name of p[i] and the list of individuals to be the
             list of domain individuals of the atom p[i] obtained from the OWL
             reasoner;
9          varRange_struct ← a variable structure with the variable name to be
             the range variable name of p[i] and the list of individuals to be the
             list of range individuals of the atom p[i] obtained from the OWL
             reasoner;
10         matrix ← AddRoleToIndividualMatrix(matrix, varDomain_struct,
             varRange_struct);
11     end
12 end
13 return matrix
```

take each of the atoms in the pattern one by one and perform iteratively (line 2 in Algorithm 13) the following:

1. If the atom is a concept atom, we create a variable structure containing two components: the variable name of the atom and the list of individuals of that atom obtained from the OWL reasoner. After that, this variable structure is passed to the function **AddConceptToIndividualMatrix** (see Section 7.3.2) to change the content of the matrix accordingly (lines 3-5 in Algorithm 13).

2. Otherwise, if the atom is a role atom, we create two variable structures, and each also contains two components: The former includes the domain variable name of the atom and the list of individuals to be obtained from domain of that atom by using the OWL reasoner, and the latter includes the range variable name of the atom and the list of individuals to be obtained from range of the atom by using the OWL reasoner. And then, these two variable structures are transferred to the function **AddRoleToIndividualMatrix** (see Section 7.3.3) to change the content of the matrix accordingly (lines 8-10 in Algorithm 13).

After calculating with the final atom of the pattern, we achieve the final individual matrix. This matrix is used to support the calculation of metrics and is returned by the algorithm.

7.3.2 The individual matrix "Add a concept atom"

The function **AddConceptToIndividualMatrix** in Algorithm 13 is represented by Algorithm 14, which is used to change the content of the input individual matrix after receiving the input information referring to an added concept atom.

Arguments of the function are an input individual matrix $matrix_{input}$ (before the change) and a variable structure var_{struct} including the variable name of the added concept atom and its list of individuals. The steps of the algorithm are performed as follows:

1. If $matrix_{input}$ is empty, we return the new individual matrix $matrix_{new}$ after adding var_{struct} to $matrix_{input}$ (lines 1-2 in Algorithm 14).

Algorithm 14: AddConceptToIndividualMatrix() : Compute the individual matrix after adding a concept atom

input : $matrix_{input}$: the input individual matrix;
var_{struct}: the variable structure of the concept atom;

output: $matrix_{new}$: a individual matrix is computed after adding the concept;

```
1  if matrix_input.isEmpty() then
2  |  matrix_new ← matrix_input.add(var_struct);
3  end
4  else
5  |  matrix_new ← all variable structures in matrix_input (keep the variable
   |    name, but set the list of individuals to be empty);
6  |  var_strconcept ← matrix_input.getStructure(var_struct.variableName());
7  |  for i = 1 ⟶ var_struct.individuals.size() do
8  |  |  for j = 1 ⟶ var_strconcept.individuals.size() do
9  |  |  |  if var_struct.individuals[i] == var_strconcept.individuals[j] then
10 |  |  |  |  foreach v_str ∈ matrix_input do
11 |  |  |  |  |  v_strtemp ← matrix_new.getStructure(v_str.variableName());
12 |  |  |  |  |  v_strtemp.add(v_str.individuals[j]);
13 |  |  |  |  |  matrix_new.replaceStructure(v_strtemp);
14 |  |  |  |  end
15 |  |  |  end
16 |  |  end
17 |  end
18 end
19 return matrix_new
```

2. Otherwise, we notice that because the rule to be obtained from the input pattern definitely satisfies the language bias, thus its atoms are transitively connected (see Definition 5). Therefore, the variable name of the added concept atom certainly exists in a certain variable structure in $matrix_{input}$. In this case, $matrix_{new}$ is created as follows:

   (a) Initialize $matrix_{new}$ from $matrix_{input}$ by copying all variable structures but set the list of individuals to be empty. This means that the variable structure in $matrix_{new}$ only contains the variable name, but its list of individuals is empty (line 5 in Algorithm 14)

   (b) Compare each individual in the input variable structure var_{struct} with each individual in the variable structure $var_{strconcept}$ which is contained in $matrix_{input}$ and the variable name of $var_{strconcept}$ is the same as the variable name of var_{struct}. If these two individuals are the same, we add all of individuals of the variable structures in the same row (index) with the individual of $var_{strconcept}$ from $matrix_{input}$ to $matrix_{new}$ (lines 6-13 in Algorithm 14).

After computing, we return $matrix_{new}$ as the output result of the algorithm.

Example 7.5. *Suppose we have individuals of an input matrix and an added concept atom as follows:*

$Matrix_{input}$			
$?x$	$?y$	$?z_1$	$?z_2$
a	b	a	b
k	f	e	c
c	b	b	g
s	a	s	b

Concept
$?y$
b
c
d

The resultant matrix of the Algorithm 14 is:

$Matrix_{output}$			
$?x$	$?y$	$?z_1$	$?z_2$
a	b	a	b
c	b	b	g

7.3.3 The individual matrix "Add a role atom"

The function **AddRoleToIndividualMatrix** in Algorithm 13 is shown by Algorithm 15, which is used to change the content and structure (if possible) of the

Algorithm 15: AddRoleToIndividualMatrix() : Compute the individual matrix after adding a role atom

input : $matrix_{input}$: the input individual matrix;
var_{struct}^{domain}: the domain variable structure of the role atom;
var_{struct}^{range}: the range variable structure of the role atom;

output: $matrix_{new}$: an individual matrix is computed after adding the role;

```
1  if matrix_input.isEmpty() then
2      matrix_new ← matrix_input.add(var_struct^domain);
3      matrix_new ← matrix_input.add(var_struct^range);
4  end
5  else
6      if var_struct^domain.variableName() ∈ matrix_input.variableNames() and
        var_struct^range.variableName() ∉ matrix_input.variableNames() then
7          matrix_new ← DomainVarExists(matrix_input, var_struct^domain, var_struct^range);
8      end
9      else if var_struct^domain.variableName() ∉ matrix_input.variableNames() and
        var_struct^range.variableName() ∈ matrix_input.variableNames() then
10         matrix_new ← RangeVarExists(matrix_input, var_struct^domain, var_struct^range);
11     end
12     else if var_struct^domain.variableName() ∈ matrix_input.variableNames() and
        var_struct^range.variableName() ∈ matrix_input.variableNames() then
13         matrix_new ← BothVarsExist(matrix_input, var_struct^domain, var_struct^range);
14     end
15     else
16         matrix_new ← BothVarsNotExist(matrix_input, var_struct^domain, var_struct^range);
17     end
18 end
19 return matrix_new
```

input individual matrix after receiving the input information referring to an added role atom.

Arguments of the function are an input individual matrix $matrix_{input}$ (before the change) and two variable structures: var_{struct}^{domain} including the domain variable name of the added role atom and its list of domain individuals; var_{struct}^{range} including the range variable name of the added role atom and its list of range individuals. The steps of the algorithm are performed as follows:

1. If $matrix_{input}$ is empty, we add both var_{struct}^{domain} and var_{struct}^{range} to the new individual matrix $matrix_{new}$ before returning $matrix_{new}$ to the function (lines 1-3 in Algorithm 15).

2. Otherwise, we must deal with one of the following four cases, depending on the variable names of the added role atom:

   (a) The domain variable name of the added role atom is in a certain variable structure of $matrix_{input}$, but the range variable name is not. This case happens when a specialized pattern has a new variable (fresh variable) in the range of the role. We invoke the function **DomainVarExists** (see Section 7.3.3.1) to compute $matrix_{input}$ in this case (lines 6-7 in Algorithm 15) and return the result to $matrix_{new}$.

   (b) The range variable name of the added role atom is in a certain variable structure of $matrix_{input}$, but the domain variable name is not. Similarly, this case happens when a specialized pattern has a new variable (fresh variable) in the domain of the role. We invoke the function **RangeVarExists** (see Section 7.3.3.2) to compute $matrix_{input}$ in this case (lines 9-10 in Algorithm 15) and return the result to $matrix_{new}$.

   (c) Both the domain variable name and the range variable name of the added role atom are in certain variable structures of $matrix_{input}$. In this case, we invoke the function **BothVarsExist** (see Section 7.3.3.3) to compute $matrix_{input}$ in this case (lines 12-13 in Algorithm 15) and return the result to $matrix_{new}$.

   (d) Both the domain variable name and the range variable name of the added role atom do not exist in any of variable structures inside $matrix_{input}$.

This case happens when we compute the confidence of the rule (Definition 11), the denominator of the formula (3.3) is calculated based on a pattern corresponding to the body of the rule, thus it is possible that the variable names of an added role atom do not exist in any of variable structures inside $matrix_{input}$.

In this case, we invoke the function **BothVarsNotExist** (see Section 7.3.3.4) to compute $matrix_{input}$ in this case (line 16 in Algorithm 15) and return the result to $matrix_{new}$.

Example 7.6. *Suppose we have the rule:*
$isMother(x, y) \leftarrow Male(y) \wedge isMother(x, z_1) \wedge isSibling(z_1, y)$

The denominator of the above rule is $Male(y) \wedge isMother(x, z_1) \wedge isSibling(z_1, y)$*, and when computing the denominator to the second atom* $isMother(x, z_1)$ *we need to call the function* ***BothVarsNotExist*** *because both variables in this second atom do not exist in* $matrix_{input}$ *(it contains the variable structure* y *at this time)*

$matrix_{new}$ is returned as the output result of the algorithm after computing.

7.3.3.1 The function DomainVarExists()

The function **DomainVarExists**, is called when the domain variable name of the added role atom is in a certain variable structure in $matrix_{input}$ while the range variable name is not, which is described in Algorithm 16.

Arguments of this function are the same as those of the function calling it in Algorithm 15. However, the domain variable name of var_{struct}^{domain} is in a certain variable structure of $matrix_{input}$, but the range variable name of var_{struct}^{range} is not. The steps of the algorithm are performed as follows:

1. Initialize $matrix_{new}$ from $matrix_{input}$ by copying all variable structures but set the list of individuals to be empty. After that, we add a variable structure including the variable name of var_{struct}^{range} along with an empty individual list to $matrix_{new}$ (lines 1-3 in Algorithm 16). Therefore, the structure of $matrix_{new}$ is different from the structure of $matrix_{input}$.

Algorithm 16: DomainVarExists()

input : $matrix_{input}$: the input individual matrix;
var^{domain}_{struct}: the domain variable structure of the role atom;
var^{range}_{struct}: the range variable structure of the role atom;

output: $matrix_{new}$: a individual matrix after being computed;

1 $matrix_{new} \leftarrow$ *all variable structures in* $matrix_{input}$ *(keep the variable name, but set the list of individuals to be empty)*;
2 $var_{newstr} \leftarrow$ *the variable structure* var^{range}_{struct} *(keep the variable name, but set the list of individuals to be empty)*;
3 $matrix_{new} \leftarrow matrix_{new}.add(var_{newstr})$;
4 $var_{strdomain} \leftarrow matrix_{input}.getStructure(var^{domain}_{struct}.variableName())$;
5 **for** $i = 1 \longrightarrow var^{domain}_{struct}.individuals.size()$ **do**
6 **for** $j = 1 \longrightarrow var_{strdomain}.individuals.size()$ **do**
7 **if** $var^{domain}_{struct}.individuals[i] == var_{strdomain}.individuals[j]$ **then**
8 **foreach** $v_{str} \in matrix_{input}$ **do**
9 $v_{strtemp} \leftarrow matrix_{new}.getStructure(v_{str}.variableName())$;
10 $v_{strtemp}.add(v_{str}.individuals[j])$;
11 $matrix_{new}.replaceStructure(v_{strtemp})$;
12 **end**
13 $v_{strtemp} \leftarrow matrix_{new}.getStructure(var^{range}_{struct}.variableName())$;
14 $v_{strtemp}.add(var^{range}_{struct}.individuals[i])$;
15 $matrix_{new}.replaceStructure(v_{strtemp})$;
16 **end**
17 **end**
18 **end**
19 **return** $matrix_{new}$

2. Compare each individual in the input domain variable structure var_{struct}^{domain} with each individual in the variable structure $var_{strdomain}$ which is contained in $matrix_{input}$ and the variable name of $var_{strdomain}$ is the same as the variable name of var_{struct}^{domain}. If these two individuals are the same, we add all of the individuals of the variable structures in the same row (index) with the individual of $var_{strdomain}$ from $matrix_{input}$ to $matrix_{new}$, and then we add an individual of var_{struct}^{range} in the same row (index) with the individual of var_{struct}^{domain} to the newly added variable structure of $matrix_{new}$ (lines 4-15 in Algorithm 16).

$matrix_{new}$ is returned as the output result of the algorithm.

Example 7.7. *Suppose we have individuals of an input matrix and an added role atom as follows:*

$Matrix_{input}$			
$?x$	$?y$	$?z_1$	$?z_2$
a	d	a	b
k	a	e	c
c	b	b	g
s	a	e	b

Role	
$?y$	$?z_3$
d	b
c	e
d	g

The resultant matrix of the Algorithm 16 is:

$Matrix_{output}$				
$?x$	$?y$	$?z_1$	$?z_2$	$?z_3$
a	d	a	b	b
a	d	a	b	g

7.3.3.2 The function RangeVarExists()

Similarily, the function **RangeVarExists** to be described in Algorithm 17 is called when the range variable name of the added role atom is in a certain variable structure in $matrix_{input}$ while the domain variable name is not.

Arguments of this function are also the same as those of the function calling it in Algorithm 15. However, the range variable name of var_{struct}^{range} is in a certain variable structure of $matrix_{input}$, but the domain variable name of var_{struct}^{domain} is not. The steps of the algorithm are performed as follows:

1. Initialize $matrix_{new}$ from $matrix_{input}$ by copying all variable structures but set the list of individuals to be empty. After that, we add a variable structure

Algorithm 17: RangeVarExists()

input : $matrix_{input}$: the input individual matrix;
var^{domain}_{struct}: the domain variable structure of the role atom;
var^{range}_{struct}: the range variable structure of the role atom;

output: $matrix_{new}$: a individual matrix after being computed;

1 $matrix_{new} \leftarrow$ *all variable structures in* $matrix_{input}$ *(keep the variable name, but set the list of individuals to be empty)*;
2 $var_{newstr} \leftarrow$ *the variable structure* var^{domain}_{struct} *(keep the variable name, but set the list of individuals to be empty)*;
3 $matrix_{new} \leftarrow matrix_{new}.add(var_{newstr})$;
4 $var_{strrange} \leftarrow matrix_{input}.getStructure(var^{range}_{struct}.variableName())$;
5 **for** $i = 1 \longrightarrow var^{range}_{struct}.individuals.size()$ **do**
6 **for** $j = 1 \longrightarrow var_{strrange}.individuals.size()$ **do**
7 **if** $var^{range}_{struct}.individuals[i] == var_{strrange}.individuals[j]$ **then**
8 **foreach** $v_{str} \in matrix_{input}$ **do**
9 $v_{strtemp} \leftarrow matrix_{new}.getStructure(v_{str}.variableName())$;
10 $v_{strtemp}.add(v_{str}.individuals[j])$;
11 $matrix_{new}.replaceStructure(v_{strtemp})$;
12 **end**
13 $v_{strtemp} \leftarrow matrix_{new}.getStructure(var^{domain}_{struct}.variableName())$;
14 $v_{strtemp}.add(var^{domain}_{struct}.individuals[i])$;
15 $matrix_{new}.replaceStructure(v_{strtemp})$;
16 **end**
17 **end**
18 **end**
19 **return** $matrix_{new}$

including the variable name of var_{struct}^{domain} along with an empty individual list to $matrix_{new}$ (lines 1-3 in Algorithm 17). Therefore, the structure of $matrix_{new}$ is different from the structure of $matrix_{input}$.

2. Compare each individual in the input range variable structure var_{struct}^{range} with each individual in the variable structure $var_{strrange}$ which is contained in $matrix_{input}$ and the variable name of $var_{strrange}$ is the same as the variable name of var_{struct}^{range}. If these two individuals are the same, we add all of individuals of the variable structures in the same row (index) with the individual of $var_{strrange}$ from $matrix_{input}$ to $matrix_{new}$, and then we add an individual of var_{struct}^{domain} in the same row (index) with the individual of var_{struct}^{range} to the newly added variable structure of $matrix_{new}$ (lines 4-15 in Algorithm 17).

$matrix_{new}$ is returned as the output result of the algorithm.

Example 7.8. *Suppose we have individuals of an input matrix and an added role atom as follows:*

$Matrix_{input}$			
$?x$	$?y$	$?z_1$	$?z_2$
a	b	a	b
k	a	e	c
c	b	b	g
s	a	e	b

Role	
$?z_3$	$?y$
b	d
e	b
g	d

The resultant matrix of the Algorithm 17 is:

$Matrix_{output}$				
$?x$	$?y$	$?z_1$	$?z_2$	$?z_3$
a	b	a	b	e
c	b	b	g	e

7.3.3.3 The function BothVarsExist()

The function **BothVarsExist** to be shown in Algorithm 18 is invoked when both the domain variable name and the range variable name of the added role atom are in certain variable structures of $matrix_{input}$.

This function also receives arguments the same as those of the function calling it in Algorithm 15. The steps of the algorithm are performed as follows:

Algorithm 18: BothVarsExist()

input : $matrix_{input}$: the input individual matrix;
var_{struct}^{domain}: the domain variable structure of the role atom;
var_{struct}^{range}: the range variable structure of the role atom;

output: $matrix_{new}$: a individual matrix after being computed;

1 $matrix_{new} \leftarrow$ *all variable structures in* $matrix_{input}$ *(keep the variable name, but set the list of individuals to be empty)*;
2 $var_{strdomain} \leftarrow matrix_{input}.getStructure(var_{struct}^{domain}.variableName())$;
3 $var_{strrange} \leftarrow matrix_{input}.getStructure(var_{struct}^{range}.variableName())$;
4 **for** $i = 1 \longrightarrow var_{struct}^{domain}.individuals.size()$ **do**
5 **for** $j = 1 \longrightarrow var_{strdomain}.individuals.size()$ **do**
6 **if** $var_{struct}^{domain}.individuals[i] == var_{strdomain}.individuals[j]$ ***and*** $var_{struct}^{range}.individuals[i] == var_{strrange}.individuals[j]$ **then**
7 **foreach** $v_{str} \in matrix_{input}$ **do**
8 $v_{strtemp} \leftarrow matrix_{new}.getStructure(v_{str}.variableName())$;
9 $v_{strtemp}.add(v_{str}.individuals[j])$;
10 $matrix_{new}.replaceStructure(v_{strtemp})$;
11 **end**
12 **end**
13 **end**
14 **end**
15 **return** $matrix_{new}$

1. Initialize $matrix_{new}$ from $matrix_{input}$ by copying all variable structures but set the list of individuals to be empty (line 1 in Algorithm 18).

2. Compare the individuals in turn on the same row (index i) of var_{struct}^{domain} and var_{struct}^{range} with the individuals on the same row (index j) of $var_{strdomain}$ and $var_{strrange}$ which are contained in $matrix_{input}$ with the condition that the variable name of $var_{strdomain}$ is the same as the variable name of var_{struct}^{domain} and the variable name of $var_{strange}$ is the same as the variable name of var_{struct}^{range}. If the individuals of var_{struct}^{domain} and $var_{strdomain}$ are the same, and the individuals of var_{struct}^{range} and $var_{strrange}$ are also the same, we add all of individuals of the variable structures in the same row (index) with both individuals of $var_{strdomain}$ and $var_{strrange}$ from $matrix_{input}$ to $matrix_{new}$ (lines 4-10 in Algorithm 18).

$matrix_{new}$ is returned as the output result of the algorithm after computing.

Example 7.9. *Suppose we have individuals of an input matrix and an added role atom as follows:*

$Matrix_{input}$			
$?x$	$?y$	$?z_1$	$?z_2$
a	b	a	b
k	a	e	c
c	b	b	g
s	a	e	b

Role	
$?x$	$?z_2$
s	b
a	b
c	a

The resultant matrix of the Algorithm 18 is:

$Matrix_{output}$			
$?x$	$?y$	$?z_1$	$?z_2$
s	a	e	b
a	b	a	b

7.3.3.4 The function BothVarsNotExist()

The function **BothVarsNotExist** to be shown in Algorithm 19 is invoked when both the domain variable name and the range variable name of the added role atom are not inside variable structures of $matrix_{input}$.

This function also receives arguments the same as those of the function calling it in Algorithm 15. The steps of the algorithm are performed as follows:

Algorithm 19: BothVarsNotExist()

input : $matrix_{input}$: the input individual matrix;
var_{struct}^{domain}: the domain variable structure of the role atom;
var_{struct}^{range}: the range variable structure of the role atom;

output: $matrix_{new}$: a individual matrix after being computed;

1 $matrix_{new} \leftarrow$ *all variable structures in* $matrix_{input}$ *(keep the variable name, but set the list of individuals to be empty)*;
2 $var_{newdomainstr} \leftarrow$ *the variable structure* var_{struct}^{domain} *(keep the variable name, but set the list of individuals to be empty)*;
3 $var_{newrangestr} \leftarrow$ *the variable structure* var_{struct}^{range} *(keep the variable name, but set the list of individuals to be empty)*;
4 $matrix_{new} \leftarrow matrix_{new}.add(var_{newdomainstr})$;
5 $matrix_{new} \leftarrow matrix_{new}.add(var_{newrangestr})$;
6 $var_{strfirst} \leftarrow matrix_{input}.getStructure(1)$;
7 **for** $i = 1 \longrightarrow var_{struct}^{domain}.individuals.size()$ **do**
8 **for** $j = 1 \longrightarrow var_{strfirst}.individuals.size()$ **do**
9 **foreach** $v_{str} \in matrix_{input}$ **do**
10 $v_{strtemp} \leftarrow matrix_{new}.getStructure(v_{str}.variableName())$;
11 $v_{strtemp}.add(v_{str}.individuals[j])$;
12 $matrix_{new}.replaceStructure(v_{strtemp})$;
13 **end**
14 $v_{strtemp} \leftarrow matrix_{new}.getStructure(var_{struct}^{domain}.variableName())$;
15 $v_{strtemp}.add(var_{struct}^{domain}.individuals[i])$;
16 $matrix_{new}.replaceStructure(v_{strtemp})$;
17 $v_{strtemp} \leftarrow matrix_{new}.getStructure(var_{struct}^{range}.variableName())$;
18 $v_{strtemp}.add(var_{struct}^{range}.individuals[i])$;
19 $matrix_{new}.replaceStructure(v_{strtemp})$;
20 **end**
21 **end**
22 **return** $matrix_{new}$

1. Initialize $matrix_{new}$ from $matrix_{input}$ by copying all variable structures but set the list of individuals to be empty. After that, we add a variable structure including the variable name of var_{struct}^{domain} along with an empty individual list and another variable structure including the variable name of var_{struct}^{range} along with an empty individual list to $matrix_{new}$ (lines 1-5 in Algorithm 19). Therefore, the structure of $matrix_{new}$ is different from the structure of $matrix_{input}$.

2. We traverse the individuals of var_{struct}^{domain} and var_{struct}^{range} in turn. Corresponding to each iteration, we add the whole individuals from $matrix_{input}$ to $matrix_{new}$, after that we fill up the two newly added variable structures with individuals of var_{struct}^{domain} and var_{struct}^{range} (lines 6-19 in Algorithm 18). This is a weakness of this algorithm as we spend a lot of memory when encountering this case.

$matrix_{new}$ is returned as the output result of the algorithm after computing.

Example 7.10. *Suppose we have individuals of an input matrix and an added role atom as follows:*

$Matrix_{input}$			
$?x$	$?y$	$?z_1$	$?z_2$
a	b	a	b
k	a	e	c
c	b	b	g

Role	
$?z_3$	$?z_4$
s	b
a	b

The resultant matrix of the Algorithm 19 is:

$Matrix_{output}$					
$?x$	$?y$	$?z_1$	$?z_2$	$?z_3$	$?z_4$
a	b	a	b	s	b
k	a	e	c	s	b
c	b	b	g	s	b
a	b	a	b	a	b
k	a	e	c	a	b
c	b	b	g	a	b

This is the Cartesian product of the two matrices.

7.4 How to compute metrics with the query optimization algorithm

As mentioned in Section 7.3, the output of the query optimization algorithm is an individual matrix used to aid in the calculation of metrics. In this section,

we based on this output matrix to compute the basic metrics (Section 3.6), the extensive metrics (Section 3.7) will be computed similarly.

7.4.1 Support

Algorithm 20: Compute support value

input : p: the pattern to be computed.

output: $value_{support}$: Support value of the rule to be obtained from p

1 $matrix_{support} \leftarrow$ *Invoke Algorithm 13 with parameter* p;
2 **if** *p[1] is ConceptAtom* **then**
3 $\quad value_{support} \leftarrow$ *count distinct individuals of the first variable structure in* $matrix_{support}$;
4 **end**
5 **else if** *p[1] is RoleAtom* **then**
6 $\quad value_{support} \leftarrow$ *count distinct individuals of the first two variable structures in* $matrix_{support}$;
7 **end**
8 **return** $value_{support}$

Algorithm 20 describes the steps for calculating the rule support (Definition 9). First, the support matrix $matrix_{support}$ is retrieved from calling the Algorithm 13 (line 1 of Algorithm 20). If the first atom of the input pattern p is a concept atom (this means that the rule r to be obtained from the pattern p has the head is the concept atom) then the returned support value is computed by counting distinct individuals of the first variable structure in $matrix_{support}$. Otherwise, If the first atom of the input pattern p is a role atom then the returned support value is computed by counting distinct individuals of the first two variable structures in $matrix_{support}$ (lines 2-8 of Algorithm 20).

Example 7.11. *Suppose we have the following output matrix of the rule r that has the head to be the concept atom:*

$Matrix_{output}$					
$?x$	$?y$	$?z_1$	$?z_2$	$?z_3$	$?z_4$
a	b	a	b	s	b
k	a	e	c	s	b
c	b	b	g	s	b
a	b	a	b	a	b
k	a	e	c	a	b
c	g	b	g	a	b

We have: $\text{supp}(r) = 3$

7.4.2 Head Coverage

Algorithm 21: Compute head coverage

input : p: the pattern to be computed.

output: $value_{HeadCoverage}$: Head coverage value of the rule to be obtained from p

1 $value_{support} \leftarrow$ *Invoke Algorithm 20 with parameter* p;
2 $head_{size} \leftarrow$ *the size of list of individuals in the atom* $\in A_f$ *whose name is the same as the name of the atom* $p[1]$ *(the first atom of* p*)*;
3 $value_{HeadCoverage} \leftarrow value_{support} / head_{size}$;
4 **return** $value_{HeadCoverage}$;

The steps for calculating the head coverage of the rule (Definition 10) are shown in Algorithm 21. First, Algorithm 20 is called to compute the support value $value_{support}$ of p (line 1 of Algorithm 21). The returned head coverage is computed by taking the quotient of $value_{support}$ with $head_{size}$ which is the size of list of individuals in the atom $\in A_f$ whose name is the same as the name of the head of the rule to be obtained from the pattern p (lines 2-4 of Algorithm 21).

Example 7.12. *Suppose we have the output matrix of the rule* r *as in the Example 7.11 and the head of* r *is the role atom whose the number of assertions in the knowledge base is 10.*

We have: $\text{supp}(r) = 4$*;* $head_{size} = 10$*; Thus* $\text{headCoverage}(r) = 4/10 = 0.4$

Algorithm 22: Compute the confidence

input : p: the pattern to be computed.

output: $value_{Confidence}$: Confidence value of the rule to be obtained from p

1 $value_{support} \leftarrow$ *Invoke Algorithm 20 with parameter* p;

2 $p_{body} \leftarrow p \setminus p[1]$;

3 $value_{denominator} \leftarrow$ *Invoke Algorithm 20 with parameter* p_{body};

4 $value_{Confidence} \leftarrow value_{support}/value_{denominator}$;

5 **return** $value_{Confidence}$;

7.4.3 Confidence

The steps for calculating the confidence of the rule (Definition 11) are presented in Algorithm 22. First, Algorithm 20 is called to compute the support value $value_{support}$ of p (line 1 of Algorithm 22). Next, we continue to apply the Algorithm 20 with the input pattern to be the body of the rule corresponding to the pattern p to compute $value_{denominator}$ (lines 2-3 of Algorithm 22). Finally, The returned rule confidence is computed by taking the quotient of $value_{support}$ with $value_{denominator}$ (lines 4-5 of Algorithm 22).

Example 7.13. *Suppose we have the output matrix of the rule r as in the Example 7.11 and the output matrix of the body of the rule r is shown below. The head of r is the role atom.*

$Matrix^{body}_{output}$					
$?x$	$?y$	$?z_1$	$?z_2$	$?z_3$	$?z_4$
a	b	a	b	s	b
k	a	e	c	s	b
c	b	b	g	s	b
a	b	a	b	a	b
k	a	e	c	a	b
c	g	b	g	a	b
c	g	a	g	a	c
c	k	a	b	b	a
b	a	c	k	a	b

We have: $\text{supp}(r) = 4$; $value_{denominator} = 6$; *Thus* $\text{conf}(r) = 4/6 \approx 0.67$

Algorithm 23: Compute the PCA-Confidence

input : p: the pattern to be computed.

output: $value_{PCAConfidence}$: PCAConfidence value of the rule to be obtained from p

1 $value_{support} \leftarrow$ *Invoke Algorithm 20 with parameter* p;
2 $atom_{head} \leftarrow p[1]$;
3 $p_{body} \leftarrow p \setminus p[1]$;
4 **if** $atom_{head}$ *is RoleAtom* **then**
5 $\quad atom_{head}.Range.Var \leftarrow$ *CreateNewFreshVariable()*;
6 $\quad p_{body} \leftarrow p_{body} \cup atom_{head}$;
7 **end**
8 $value_{denominator} \leftarrow$ *Invoke Algorithm 20 with parameter* p_{body};
9 $value_{PCAConfidence} \leftarrow value_{support}/value_{denominator}$;
10 **return** $value_{PCAConfidence}$;

7.4.4 PCA-Confidence

The steps for calculating the confidence of the rule (Definition 12) are presented in Algorithm 23. First, Algorithm 20 is called to compute the support value $value_{support}$ of p (line 1 of Algorithm 23). If the head of the rule to be obtained from the input pattern p is the concept atom, the returned pca-confidence of the rule is computed as the confidence. On the contrary, if the head is the role atom, we change the range variable of the head to a new variable that does not exist inside p and then move that atom (the head) in the first position to the last position. After that, we continue to apply the Algorithm 20 with the modified patten to compute $value_{denominator}$. The returned rule pca-confidence is computed by taking the quotient of $value_{support}$ with $value_{denominator}$ (lines 4-10 of Algorithm 23).

7.5 Evaluation

We carry out a comparison of the query optimization algorithm and SPARQL-DL queries based on execution time. Both solutions have been applied on both the level-wise generate-and-test algorithm (Chapter 4) and the evolutionary algorithm (Chapter 5) along with the inputs which are publicly available ontologies (Table 4.1).

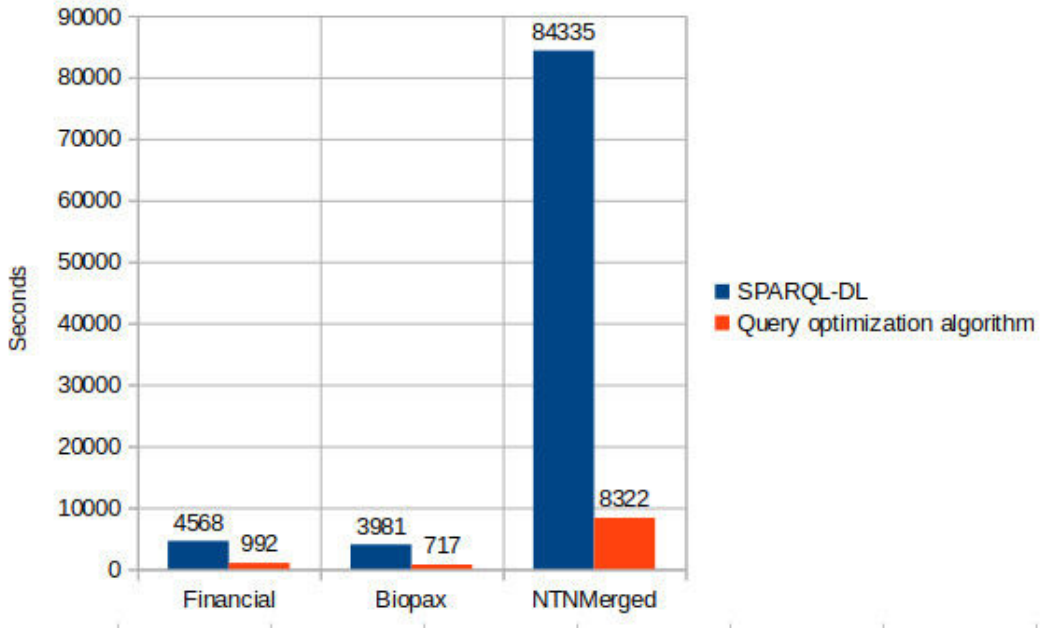


Figure 7.2: Execution time of the Level-wise Generate-And-Test algorithm

All experiments have been performed on a Dell computer with Intel Core i5-4210U CPU at 1.7GHz x 4, 8GB RAM, under the Ubuntu 16.04 LTS 64-bit operating system.

1. **Applying to level-wise generate-and-test algorithm**

   The algorithm sets the input parameters as in Section 4.4.2. Query optimization algorithm or SPARQL-DL queries are applied to compute metrics Head Coverage (line 5 in Algorithm 5) and Rule Confidence (line 8 in Algorithm 5) of each generated pattern. The chart in Figure. 7.2 shows that the execution time of the query optimization algorithm is better than the SPARQL-DL query in all three ontologies.

2. **Applying to evolutionary algorithm**

   In this algorithm, the input parameters are set as in Section 5.4. Query optimization algorithm or SPARQL-DL queries are applied to compute the fitness value of each individual within the population. The chart in Figure. 7.3 presents that the execution time of the query optimization algorithm is still better than the SPARQL-DL query in all three ontologies. In the ontology NTNMerged, the execution time of SPARQL-DL query is not acceptable due

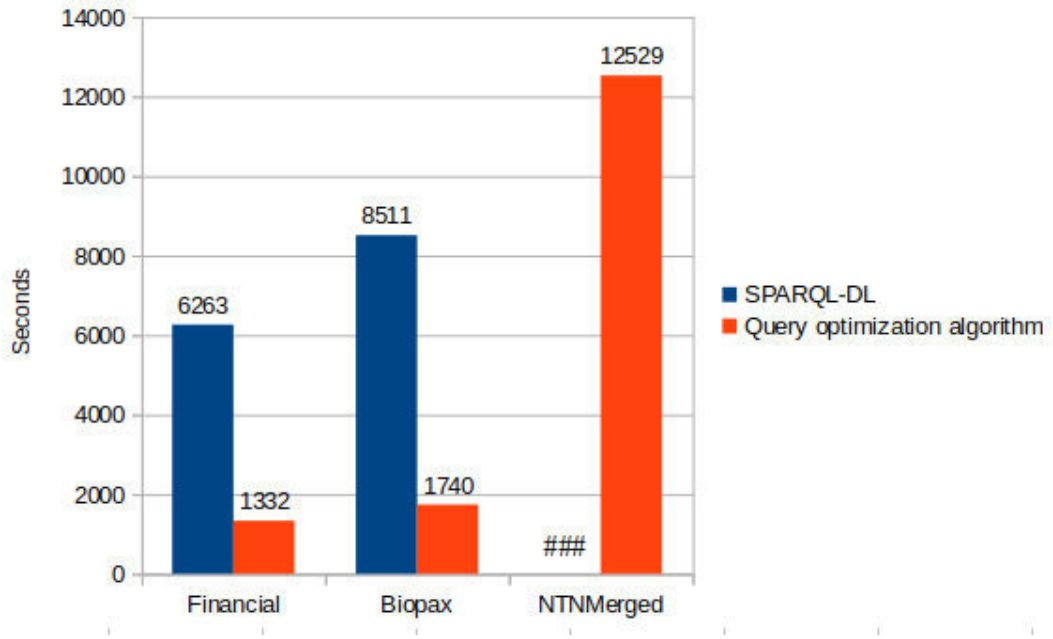


Figure 7.3: Execution time of the Evolutionary algorithm

to it taking too long, the symbol ### is used to express this.

Chapter 8

Conclusions and perspectives

Contents

8.1 Conclusions 127

8.1.1 Discovering hidden knowledge patterns 128

8.1.2 Selecting the best asymmetric metrics 129

8.1.3 Improving computing speed 130

8.2 Future Work 131

8.1 Conclusions

OWL ontologies are one of the key foundations of the Semantic Web context, which describe formally shared conceptualizations of a domain. Meanwhile, LOD is the first massive deployment of the Semantic Web, which contains ontologies in it. Therefore, knowledge extraction and analysis from ontologies are obvious demands.

In this thesis, we have proposed algorithms for discovering hidden knowledge patterns from ontological knowledge bases, in the form of multi-relational association rules coded in SWRL, by exploiting the evidence coming from the assertional data. We hope that this research result will be used for enriching an ontology both at terminological (schema) and assertional (facts) level, even in presence of incompleteness and/or noise. Our approach in the thesis consists of the following steps:

1. Building algorithms to discover hidden knowledge patterns from ontological knowledge bases.
2. Improving metrics applied in the algorithms.
3. Improving computing speed.

8.1.1 Discovering hidden knowledge patterns

We built the two following algorithms to discover hidden knowledge patterns coded in SWRL from ontological knowledge bases:

1. The first algorithm, a level-wise generate-and-test approach (Chapter 4), discovers all possible SWRL rules in the given space of rules that is limited by a maximum length of rule. In this algorithm, first, we used an downward refinement operator (Section 4.2) to generate knowledge patterns respecting the language bias (Section 3.5), after that we employed the basic metrics (Section 3.6) to evaluate the quality of knowledge patterns. Besides, a heuristic was developed to avoid generating rules that have the same semantics (Algorithm 5). Finally, SWRL rules to be discovered are the rules obtained from knowledge patterns by setting the first atom as the head of the rule and the remaining as the rule body. In addition, the rules to be discovered are not allowed to contradict the reference knowledge base(Section 4.3.1.3), this ensures that the discovered rules are easily integrated within the ontology. The advantage of this approach is that we might obtain all discovered rules in a given space and ensure them to be consistent with the knowledge base. However, its disadvantage is that it is difficult to reach the discover long rules because of performance problem (slow execution time).

2. The second algorithm, an evolutionary approach (Chapter 5), has the same objective as the first algorithm but is proposed to overcome its disadvantages. The algorithm maintains a population of the best knowledge patterns and makes it evolve by iteratively applying genetic operators (Section 5.3). These genetic operators are designed to create offspring (knowledge patterns) from their parents selected in the population and these offspring must respect the language bias (Section 3.5); offspring are then added back into the population to compete directly with old individuals. This ensures that individuals (knowledge patterns) in the final generation are the best individuals in the whole traversal process. A fitness function based on metrics of pattern (Section 3.6), is used to assist the operators in the evolutionary algorithm. Discovered SWRL rules to be extracted from the final population are knowledge patterns that satisfy two conditions: (i) Their fitness value is higher than

a given threshold and (ii) They do not contradict the knowledge base. The advantage of this approach is that it is possible to face larger search spaces than with the first algorithm. This means that we can discover rules having a maximum length greater than the length of the rules discovered by first algorithm without worrying about performance problems. The discovered rules to be generated by this algorithm are also easily integrated with in a knowledge base. The disadvantage of this approach is that it can miss some rules; however, in order to overcome this disadvantage, we might increase the number of generations for the evolutionary algorithm accordingly.

In order to compare the quality of solutions, we based ourselves on the number of discovered rules and used the metrics presented in Section 3.8 to evaluate rule precision. AMIE [GTHS13] is a state-of-the-art system outperforming others state-of-the-art ILP systems in terms of the number of discovered rules. We compared our system with AMIE using the same samples of the ontologies and found that the number of rules discovered by both of our algorithms was greater than the number of rules discovered by AMIE (Table 5.3). For the ability to predict assertional knowledge (Sections 4.4.2.1 and 5.4.1), both above algorithms indicate that the discovered rules are able to predict new assertional knowledge (*match rate* > 0), no contradicting knowledge is predicted and show the ability to exploit accurate rules (*commission error rate* $= 0$), come up with rules that induce previously unknown facts, this means that new knowledge is found which is not logically derivable (*induction rate* > 0). In addition, the second algorithm creates a very large number of predictions compared to the first algorithm. Through experiments (Sections 4.4 and 5.4), we found that the evolutionary approach is currently the best approach to discovering hidden knowledge patterns coded in SWRL from ontological knowledge bases.

8.1.2 Selecting the best asymmetric metrics

Language bias is a set of constraints that the generated rules must respect in order to avoid to create redundant or trivial rules. However, to ensure quality for the generated rules, we employ metrics to evaluate them. Besides the basic metrics that we have described (Section 3.6) and used (Chapter 4 and 5), there are other alternative metrics (Section 3.7) proposed for assessing the quality of the generated

rules. In this thesis, we only used asymmetric metrics because values of an asymmetric metric for $head \longrightarrow body$ and $body \longrightarrow head$ may not be the same. These asymmetric metrics have originally been proposed for scoring association rules.

Because there are many asymmetric metrics (Table 6.1) to evaluate the quality of the rules, we have decided to compare these metrics to select the best metrics that are suitable for the semantic web. Our comparative goal was based on the number of discovered rules and used metrics in Section 3.8 to assess rule precision. Asymmetric metrics were applied in turn to evolutionary algorithm (shown in Chapter 5) as a fitness function.

By observing the results of the experiment (Section 6.2), we concluded that *HeadCoverage*, *Confidence*, *PCA-Confidence*, *Certainty Factor*, and *Added Value* are the best metrics because they discover the largest number of rules (see Table 6.2) and come up with rules that induce a large number of previously unknown facts, with a very large absolute number of correct predictions. In addition, when compared to the state-of-the-art system AMIE, the best metrics also discover more rules than AMIE. However, these best metrics do not outperform the second fitness funtion in Section 5.3.3 ($f_{HPCA}(r) = HeadCoverage(r) + PCAConfidence(r)$). We observed that this fitness function is a combination of two metrics, thus this will be a new promising direction of research.

8.1.3 Improving computing speed

In order to compute the metrics (Sections 3.6 and 3.7) for a rule expressed in the form of SWRL, we might perform querying on SPARQL-DL (see Section 7.2.1) because SPARQL-DL query statement is easily converted from the format of SWRL. However, the data query speed of SPARQL-DL is rather slow, thus it is not suitable for computing constantly in a short time. Therefore, we came up with an algorithm in Section 7.3 to compute metrics with the goal of replacing the SPARQL-DL query to speed up the rule evaluation process.

The algorithm we propose is based on the hash join algorithm used in the implementation of RDBMS (Relational Database Management System) but is adapted to fit the notions in the semantic web. The input of the algorithm is a SWRL rule and an ontology used to retrieve information. The output of the algorithm is a data structure employed to support the computation of metrics.

Reviews and comparisons in Section 7.5 show that the processing speed of the algorithm is much faster than the SPARQL-DL query. Therefore, this is a direction that we might follow to improve the performance of the system. However, the disadvantage of this algorithm is that it must store all the calculated data in memory, which means that it uses a significant amount of memory for a large amount of data. Thus, the proposed algorithm is only suitable for small and medium ontologies. In order to ensure the performance of the system and work with big data, we need to find a new approach that is more appropriate and combine it with the proposed algorithm.

8.2 Future Work

In this thesis, we have proposed some approaches based on ideas taken from Inductive Logic Programming to discover hidden knowledge patterns from ontological knowledge bases, in the form of multi-relational association rules expressed in SWRL. The results of the thesis have achieved some positive effects and can be applied to the ontological knowledge bases in practice, as it is evidenced by the experimental results. In the future, we will continue to research and improve the algorithm in order to achieve better performance. Specifically, future work might focus on the main following aspects:

- **Exploration of various possible combinations of the promising metrics**: We have compared the popular asymmetric metrics in Chapter 6 and identified the best five metrics for further exploration. However, when running the experiment with a fitness function combining two metrics, we found that this combined fitness function performed better than each of the best five metrics individually. Therefore, the idea of finding a combined fitness functions has been proposed for future work.

- **Development of other metrics suited for scenarios based on OWA (Open World Assumption)**: There are currently many metrics used to evaluate the quality of the association rules, but most of them support for scenarios based on CWA (Closed World Assumption). Only very few metrics are employed to assist the scenario based on OWA which is suitable with the ontological knowledge bases. However, their effectiveness is still not what we

would expect. Therefore, another work proposed in the future is to research and develop metrics suited for OWA.

– **Scalability**: In order to bring research results into practice, our next work will go toward applying our solutions in datasets from Linked Open Data (LOD). By using these datasets, we will continue to improve the algorithm to process more incomplete or unusual dataset. In addition, that is also a way to increase the reliability of our solution in practice.

– **Parallelization according to programming models**: As mentioned in Section 8.1.3, we need to find a new solution combining with the results of this research to overcome weaknesses and be able to work with big data. One future direction along which we aim to expand our study is to parallelize the algorithms. We will perform in parallel by convert the algorithms presented in the thesis to program models such MapReduce to take advantage of frameworks like Hadoop or Spark, in order to be able to perform big data analytics.

Appendix A

The algorithm for generating a closed rule

A.1 An algorithm description

The algorithm to be mentioned in this part focuses on creating a closed rule from some input arguments. Specifically, we have an input pattern p and an input atom A, where p is a closed rule or contains at most two fresh variables (a fresh variable appears only once in the pattern) and A is an atom attached to the tail of p. The output of the algorithm is the atom A adjusted variables so that the rule r_{output} corresponding to the pattern $\{p \cup A\}$ satisfying one of the following conditions:

1. r_{output} is a closed rule.
2. if r_{output} has a length of two atoms then it contains at most two fresh variables.
3. if r_{output} has a length greater than two atoms then it contains at most one fresh variables.

Algorithm 24 represents the whole above idea and is named AdjustAtom(). The algorithm is divided into two smaller functions that depend on the type of atom whose variables need to be adjusted. To be more specific, if A is a concept atom, then the value of the output A will be received from the function AdjustConceptAtom() in Algorithm 25 (the lines 1-2 in Algorithm 24), otherwise if A is a role atom then the value of the output A will be received from the function AdjustRoleAtom() in Algorithm 29 (the lines 4-5 in Algorithm 24).

A.1.1 The adjusted atom is a concept atom

Algorithm 25 returns a concept atom whose variable is adjusted based on the input pattern p, so that the rule corresponding to the pattern $\{p \cup C\}$ satisfies one of the

Algorithm 24: The function AdjustAtom(): Adjust the variables in the atom to ensure the language bias is respected

input : p: the input pattern; A: the atom needs to be adjusted variables;
output: A: The atom have been adjusted variables based on p

```
1 if A is ConceptAtom then
2 |   A ← AdjustConceptAtom(p, A) in Alg 25;
3 end
4 else if A is RoleAtom then
5 |   A ← AdjustRoleAtom(p, A) in Alg 29;
6 end
7 return A
```

conditions in Section A.1. There are three cases occurring inside this algorithm:

1. $p = \emptyset$: In this case, p does not contain any atoms, thus the variable of the output concept atom receives the given default value (the lines 1-2 in Algorithm 25).

2. **The length of** p **is** 1: In this case, the variable in the output concept atom will receive one of the variables already exists in p. If the head of the rule (the first element of the pattern p) corresponding to the pattern p is the concept atom then the value of the output C will be received from the function AdjustConceptAtomWithP1AndPHeadIsConceptAtom() in Algorithm 26 (the lines 5-6 in Algorithm 25); otherwise if the head is the role atom then the variable of the output concept atom will randomly pick one of the two variables of the head (the lines 8-9 in Algorithm 25).

3. **The length of** p **is greater than** 1: In this case, the variable in the output concept atom will also receive one of the variables already existing in p. If the head of the rule corresponding to the pattern p is a concept atom then the value of the output C will be received from the function AdjustConceptAtomWithPHeadIsConceptAtom() in Algorithm 27 (the lines 13-14 in Algorithm 25); otherwise if the head is a role atom, then the value of the output C will be received from the function AdjustConceptAtomWithPHeadIsRoleAtom() in Algorithm 28 (the lines 16-17 in Algorithm 25).

Algorithm 25: The function AdjustConceptAtom(): Adjust the variable in the concept atom to ensure the language bias is respected

input : p: the input pattern; C: the concept atom needs to be adjusted variables;

output: C: The concept atom have been adjusted variable based on p

```
if p = ∅ then
    C.Var ← The given default variable of concept atom;
end
else if p.length() = 1 then
    if p.headAtom is ConceptAtom then
        C ← AdjustConceptAtomWithP1AndPHeadIsConceptAtom(p, C) in
          Alg 26;
    end
    else if p.headAtom is RoleAtom then
        C.Var ← randomly pick between two variables
          p.headAtom.DomainVar and p.headAtom.RangeVar;
    end
end
else if p.length() > 1 then
    if p.headAtom is ConceptAtom then
        C ← AdjustConceptAtomWithPHeadIsConceptAtom(p, C) in Alg 27;
    end
    else if p.headAtom is RoleAtom then
        C ← AdjustConceptAtomWithPHeadIsRoleAtom(p, C) in Alg 28;
    end
end
return C
```

– **The length of p is 1 and the head is the concept atom (Algorithm 26)**

Algorithm 26: The function AdjustConceptAtomWithP1AndPHeadIsConceptAtom(): Adjust the variables in the concept atom to ensure the language bias is respected with p.length()=1 and p.headAtom is a concept atom

input : p: the input pattern; C: the concept atom needs to be adjusted variables;
a global variable A_f: a list of frequent atoms;

output: C: The concept atom have been adjusted variable based on p

```
1  Af_temp ← A_f;
2  returnAtom ← false;
3  repeat
4      if ¬C.isSubsumedBy(p.headAtom) then
5          C.Var ← p.headAtom.Var;
6          returnAtom ← true;
7      end
8      else
9          Af_temp ← Af_temp \ C;
10         C ← pick an concept atom in Af_temp at random;
11     end
12 until returnAtom = true;
13 return C
```

This function (Algorithm 26) returns the concept atom whose variable will receive the variable of the head if the concept atom is not subsumed by the head (the lines 4-5 in Algorithm 26); otherwise a new concept atom will be picked at random from the list A_f until the picked concept atom is not subsumed by the head, after that its variable will receive the variable of the head (the lines 1-12 in Algorithm 26).

– **The length of p is greater than 1 and the head is a concept atom (Algorithm 27)**

Algorithm 27: The function AdjustConceptAtomWithPHeadIsConceptAtom(): Adjust the variables in the concept atom to ensure the language bias is respected with p.length()>1 and p.headAtom is a concept atom

input : p: the input pattern; C: the concept atom needs to be adjusted variables;
$concepts_p^{body}$: concept names appearing in the body of the pattern;
$roles_p^{body}$: role names appearing in the body of the pattern;
$vars_p$: variable names appearing in the pattern p;

output: C: The concept atom have been adjusted variable based on p

1 **if** $C.isSubsumedBy(p.headAtom)$ **then**
2 $\quad vars_p \leftarrow vars_p \setminus p.headAtom.Var$;
3 **end**
4 $subC_p = concepts_p^{body}.getConceptsSubsumedBy(C)$;
5 $superC_p = concepts_p^{body}.getConceptsSubsuming(C)$;
6 $subR_p^{Domain} = roles_p^{body}.getRolesWithDomainSubsumedBy(C)$;
7 $subR_p^{Range} = roles_p^{body}.getRolesWithRangeSubsumedBy(C)$;
8 $used \leftarrow subC_p.getVars() \cup superC_p.getVars() \cup subR_p^{Domain}.getVars() \cup subR_p^{Range}.getVars()$;
9 $vars_p \leftarrow vars_p \setminus used$;
10 $C.Var \leftarrow$ *pick a variable in* $vars_p$ *at random*;
11 **return** C

This function (Algorithm 27) returns the concept atom C whose variable is randomly picked in the variables appearing in the pattern p after removing the unsuitable variables as follows:

* The variable in the head will be removed if C is subsumed by the head (the lines 1-2 in Algorithm 27).
* The variables of the concept atoms in the body will be removed if C subsumes or is subsumed by these concept atoms (the lines 4-5 in Algorithm 27).

* The variables of the role atoms in the body will be removed if the domain or the range of these role atoms are subsumed by C (the lines 6-7 in Algorithm 27).

– **The length of p is greater than 1 and the head is a role atom (Algorithm 28)**

Algorithm 28: The function AdjustConceptAtomWithPHeadIsRoleAtom(): Adjust the variables in the concept atom to ensure the language bias is respected with p.length()>1 and p.headAtom is a role atom

---

input : p: the input pattern; C: the concept atom needs to be adjusted variables;
$concepts_p^{body}$: concept names appearing in the body of the pattern;
$roles_p^{body}$: role names appearing in the body of the pattern;
$vars_p$: variable names appearing in the pattern p;

output: C: The concept atom have been adjusted variable based on p

1 $subC_p = concepts_p^{body}.getConceptsSubsumedBy(C)$;
2 $superC_p = concepts_p^{body}.getConceptsSubsuming(C)$;
3 $subR_p^{Domain} = roles_p^{body}.getRolesWithDomainSubsumedBy(C)$;
4 $subR_p^{Range} = roles_p^{body}.getRolesWithRangeSubsumedBy(C)$;
5 $used \leftarrow subC_p.getVars() \cup superC_p.getVars() \cup subR_p^{Domain}.getVars() \cup subR_p^{Range}.getVars()$;
6 $vars_p \leftarrow vars_p \setminus used$;
7 $C.Var \leftarrow$ *pick a variable in* $vars_p$ *at random*;
8 **return** C

This function (Algorithm 28) returns the concept atom C whose variable is randomly picked in the variables appearing in the pattern p after removing the unsuitable variables as follows:

* The variables of the concept atoms in the body will be removed if C subsumes or is subsumed by these concept atoms (the lines 1-2 in Algorithm 28).

* The variables of the role atoms in the body will be removed if the domain or the range of these role atoms are subsumed by C (the lines 3-4 in Algorithm 28).

A.1.2 The adjusted atom is a role atom

Algorithm 29 returns a role atom whose variables are adjusted based on the input pattern p so that the rule corresponding to the pattern $\{p \cup R\}$ satisfies one of the conditions in Section A.1. There are three cases occurring inside this algorithm:

1. $p = \emptyset$: In this case, p does not contain any atoms, thus the domain and range variables of the output role atom receive the given default value (the lines 1-3 in Algorithm 29).

2. **The length of p is** 1: In this case, if the head of the rule corresponding to the pattern p is the concept atom then the value of the output R will be received from the function AdjustRoleAtomWithP1AndPHeadIsConceptAtom() in Algorithm 30 (the lines 6-7 in Algorithm 29); otherwise if the head is the role atom then the value of the output R will be received from the function AdjustRoleAtomWithP1AndPHeadIsRoleAtom() in Algorithm 31 (the lines 9-10 in Algorithm 29).

3. **The length of p is greater than** 1: In this case, if the head of the rule corresponding to the pattern p is the concept atom then the value of the output R will be received from the function AdjustRoleAtomWithPHeadIsConceptAtom() in Algorithm 32 (the lines 14-15 in Algorithm 29); otherwise if the head is the role atom then the value of the output R will be received from the function AdjustRoleAtomWithPHeadIsRoleAtom() in Algorithm 35 (the lines 17-18 in Algorithm 29).

– **The length of p is** 1 **and the head is a concept atom (Algorithm 30)**

This function (Algorithm 30) returns the role atom whose variable names will be identified as follows:

* The domain variable receives the variable of the head and the range

Algorithm 29: The function AdjustRoleAtom(): Adjust the variables in the role atom to ensure the language bias is respected

input : p: the input pattern; R: the role atom needs to be adjusted variables;

output: R: The role atom have been adjusted variables based on p

```
1  if p = ∅ then
2      R.DomainVar ← The given default domain variable of concept atom;
3      R.RangeVar ← The given default range variable of concept atom;
4  end
5  else if p.length() = 1 then
6      if p.headAtom is ConceptAtom then
7          R ← AdjustRoleAtomWithP1AndPHeadIsConceptAtom(p, R) in
             Alg 30;
8      end
9      else if p.headAtom is RoleAtom then
10         R ← AdjustRoleAtomWithP1AndPHeadIsRoleAtom(p, R) in Alg 31;
11     end
12 end
13 else if p.length() > 1 then
14     if p.headAtom is ConceptAtom then
15         R ← AdjustRoleAtomWithPHeadIsConceptAtom(p, R) in Alg 32;
16     end
17     else if p.headAtom is RoleAtom then
18         R ← AdjustRoleAtomWithPHeadIsRoleAtom(p, R) in Alg 35;
19     end
20 end
21 return R
```

Algorithm 30: The function AdjustRoleAtomWithP1AndPHeadIsConceptAtom(): Adjust the variables in the role atom to ensure the language bias is respected with p.length()=1 and p.headAtom is a concept atom

input : p: the input pattern; R: the role atom needs to be adjusted variables;
a global variable A_f: a list of frequent atoms;

output: R: The role atom have been adjusted variables based on p

$Af_{temp} \leftarrow A_f$;
$returnAtom \leftarrow false$;
$var_{head} \leftarrow p.headAtom.Var$;
$z \leftarrow CreateNewFreshVariable()$;
repeat
  if $\neg R.Domain.isSubsumedBy(p.headAtom)$ *and* $R.Range.isSubsumedBy(p.headAtom)$ **then**
    $R.Domain.Var \leftarrow var_{head}$; $R.Range.Var \leftarrow z$; $returnAtom \leftarrow true$;
  end
  else if $R.Domain.isSubsumedBy(p.headAtom)$ *and* $\neg R.Range.isSubsumedBy(p.headAtom)$ **then**
    $R.Domain.Var \leftarrow z$; $R.Range.Var \leftarrow var_{head}$; $returnAtom \leftarrow true$;
  end
  else if $\neg R.Domain.isSubsumedBy(p.headAtom)$ *and* $\neg R.Range.isSubsumedBy(p.headAtom)$ **then**
    $R.Domain.Var \leftarrow$ *randomly pick between two variables* var_{head} *and* z;
    $R.Range.Var \leftarrow (var_{head} \cup z) \setminus R.Domain.Var$;
  end
  else
    $Af_{temp} \leftarrow Af_{temp} \setminus R$;
    $R \leftarrow$ *pick an role atom in* Af_{temp} *at random*;
  end
until $returnAtom = true$;
return R

variable receives a new fresh variable if the range of R is subsumed by the head but the domain of R is not (the lines 6-7 in Algorithm 30).

* The range variable receives the variable of the head and the domain variable receives a new fresh variable if the domain of R is subsumed by the head but the range of R is not (the lines 9-10 in Algorithm 30).
* The domain variable receives the variable of the head or a new fresh variable, the range variable receives the remaining variable if both the domain and the range of R are not subsumed by the head (the lines 12-14 in Algorithm 30).
* In the last case, a new role atom will be picked at random from the list A_f until the picked role atom satisfies the above conditions, after that its variables will be received according to the satisfied condition (the lines 1-20 in Algorithm 30).

– **The length of p is 1 and the head is a role atom (Algorithm 31)**

This function (Algorithm 31) returns the role atom whose variable names will be identified by one of the following three variables: a new fresh variable, the domain and the range variables of the head (the lines 1-5 in Algorithm 31). If the position of variables in R coincides with the position of variables in the head and R is SubProperty of the head then we will change the position of variables in R (the lines 6-10 in Algorithm 31).

– **The length of p is greater than 1 and the head is a concept atom (Algorithm 32)**

This function (Algorithm 32) returns the role atom R whose variables is received after removing the unsuitable variables as follows:

* The variable in the head will be removed from the list of variables of domain if the domain of R is subsumed by the head (the lines 3-4 in Algorithm 32).
* The variable in the head will be removed from the list of variables of range if the range of R is subsumed by the head (the lines 6-7 in Algorithm 32).

Algorithm 31: The function AdjustRoleAtomWithP1AndPHeadIsRoleAtom(): Adjust the variables in the role atom to ensure the language bias is respected with p.length()=1 and p.headAtom is a role atom

input : p: the input pattern; R: the role atom needs to be adjusted variables;
$vars_p$: variable names appearing in the pattern p;

output: R: The role atom have been adjusted variables based on p

$z \leftarrow$ *CreateNewFreshVariable()*;
$vars_p \leftarrow vars_p \cup z$;
$var_{domain} \leftarrow$ *pick a variable in* $vars_p$ *at random*;
$vars_p \leftarrow vars_p \setminus var_{domain}$;
$var_{range} \leftarrow$ *pick a variable in* $vars_p$ *at random*;
if *(*var_{domain} *= p.headAtom.Domain.Var) and (*var_{range} *= p.headAtom.Range.Var)* **then**
    if *R.isSubPropertyOf(p.headAtom)* **then**
        $var_{temp} \leftarrow var_{domain}$;
        $var_{domain} \leftarrow var_{range}$;
        $var_{range} \leftarrow var_{temp}$;
    end
end
R.Domain.Var $\leftarrow var_{domain}$;
R.Range.Var $\leftarrow var_{range}$;
return R

Algorithm 32: The function AdjustRoleAtomWithPHeadIsConceptAtom(): Adjust the variables in the role atom to ensure the language bias is respected with p.length()>1 and p.headAtom is a concept atom

input : p: the input pattern; R: the role atom needs to be adjusted variables;
$vars_p$: variable names appearing in the pattern p;

output: R: The role atom have been adjusted variables based on p

$vars_{domain} \leftarrow vars_p$;
$vars_{range} \leftarrow vars_p$;
if *R.Domain.isSubsumedBy(p.headAtom)* **then**
  $vars_{domain} \leftarrow vars_{domain} \setminus p.headAtom.Var$;
end
if *R.Range.isSubsumedBy(p.headAtom)* **then**
  $vars_{range} \leftarrow vars_{range} \setminus p.headAtom.Var$;
end
$subDomainR_p = concepts_p^{body}.getConceptsSubsuming(R.Domain)$;
$subRangeR_p = concepts_p^{body}.getConceptsSubsuming(R.Range)$;
$vars_{domain} \leftarrow vars_{domain} \setminus subDomainR_p.Vars$;
$vars_{range} \leftarrow vars_{range} \setminus subRangeR_p.Vars$;
if $\neg p.isExistFreshVariable()$ **then**
  $R_{temp} \leftarrow NoFreshVariablesExist()$ in Alg 33;
end
else
  $R_{temp} \leftarrow FreshVariablesExist()$ in Alg 34;
end
$R.Domain.Var \leftarrow R_{temp}.Domain.Var$;
$R.Range.Var \leftarrow R_{temp}.Range.Var$;
return R

* The variables of the concept atoms in the body will be removed from the list of variables of domain or range if these concept atoms subsume R.Domain or R.Range respectively (the lines 9-12 in Algorithm 32).

* If the pattern p does not contains the fresh variable then the value of the output R will be received from the function NoFreshVariablesExist() in Algorithm 33; otherwise if the pattern p contains the fresh variable then the value of the output R will be received from the function FreshVariablesExist() in Algorithm 34 (the lines 13-20 in Algorithm 32).

(a) **The function NoFreshVariablesExist()**

This function (Algorithm 33) has the input containing two lists of domain and range variables and returns the role atom R whose variables will be identified as follows:

* If the size of the list of domain variables is greater than one and the size of the list of range variables equals 0 then the domain variable of R is picked randomly in the list of domain variables, the range variable of R receives a new fresh variable (the lines 1-3 in Algorithm 33).

* If the size of the list of range variables is greater than one and the size of the list of domain variables equals 0 then the range variable of R is picked randomly in the list of range variables, the domain variable of R receives a new fresh variable (the lines 5-7 in Algorithm 33).

* If both size of the lists of domain and range variables is greater than one then add a new fresh variable to both lists, after that select randomly a pair of variables in two lists of domain and range variables such that this pair of variables is different from pairs of variables of role atoms in $roles_p^{body}$ that are SubProperty or SuperProperty of R (the lines 9-13 in Algorithm 33).

(b) **The function FreshVariablesExist()**

This function (Algorithm 34) has the input containing two lists of domain and

Algorithm 33: The function NoFreshVariablesExist(): Support the function AdjustRoleAtomWithP-HeadIsConceptAtom() to select the domain and range variables accordingly in case no fresh variables exist

input : R: the role atom needs to be adjusted variables;
$vars_{domain}$: the list of domain variables;
$vars_{range}$: the list of range variables;
$roles_p^{body}$: role names appearing in the body of the pattern;

output: R: The role atom have been adjusted variables

if *($vars_{domain}$.size() > 0) and ($vars_{range}$.size() = 0)* **then**
  $var_{domain} \leftarrow$ *pick a variable in* $vars_{domain}$ *at random*;
  $var_{range} \leftarrow$ *CreateNewFreshVariable()*;
end
else if *($vars_{domain}$.size() = 0) and ($vars_{range}$.size() > 0)* **then**
  $var_{domain} \leftarrow$ *CreateNewFreshVariable()*;
  $var_{range} \leftarrow$ *pick a variable in* $vars_{range}$ *at random*;
end
else if *($vars_{domain}$.size() > 0) and ($vars_{range}$.size() > 0)* **then**
  $z \leftarrow$ *CreateNewFreshVariable()*;
  $vars_{domain} \leftarrow vars_{domain} \cup z$;
  $vars_{range} \leftarrow vars_{range} \cup z$;
  Select randomly a pair of variables $var_{domain} \in vars_{domain}$ *and* $var_{range} \in vars_{range}$ *such that this pair of variables is different from pairs of variables of role atoms in* $roles_p^{body}$ *that are SubProperty or SuperProperty of* R;
end
$R.Domain.Var \leftarrow var_{domain}$;
$R.Range.Var \leftarrow var_{range}$;
return R

Algorithm 34: The function FreshVariablesExist(): Support the function AdjustRoleAtomWithPHeadIsConceptAtom() to select the domain and range variables accordingly in case there are fresh variables

input : R: the role atom needs to be adjusted variables;
$vars_{domain}$: the list of domain variables;
$vars_{range}$: the list of range variables;
$roles_p^{body}$: role names appearing in the body of the pattern;

output: R: The role atom have been adjusted variables

if *($vars_{domain}$.size() > 0) and ($vars_{range}$.size() = 0)* **then**
  $vars_{domain} \leftarrow vars_{domain} \setminus$ *GetFreshVariable()*;
  $var_{domain} \leftarrow$ *pick a variable in* $vars_{domain}$ *at random*;
  $var_{range} \leftarrow$ *GetFreshVariable()*;
end
else if *($vars_{domain}$.size() = 0) and ($vars_{range}$.size() > 0)* **then**
  $vars_{range} \leftarrow vars_{range} \setminus$ *GetFreshVariable()*;
  $var_{domain} \leftarrow$ *GetFreshVariable()*;
  $var_{range} \leftarrow$ *pick a variable in* $vars_{range}$ *at random*;
end
else if *($vars_{domain}$.size() > 0) and ($vars_{range}$.size() > 0)* **then**
  Select randomly a pair of variables $var_{domain} \in vars_{domain}$ *and* $var_{range} \in vars_{range}$ *such that this pair of variables is different from pairs of variables of role atoms in* $roles_p^{body}$ *that are SubProperty or SuperProperty of* R;
end
$R.Domain.Var \leftarrow var_{domain}$;
$R.Range.Var \leftarrow var_{range}$;
return R

range variables and returns the role atom R whose variables will be identified as follows:

* If the size of the list of domain variables is greater than one and the size of the list of range variables equals 0 then the domain variable of R is picked randomly in the list of domain variables after removing the fresh variable, the range variable of R receives a new fresh variable (the lines 1-4 in Algorithm 34).
* If the size of the list of range variables is greater than one and the size of the list of domain variables equals 0 then the range variable of R is picked randomly in the list of range variables after removing the fresh variable, the domain variable of R receives a new fresh variable (the lines 6-9 in Algorithm 34).
* If both size of the lists of domain and range variables is greater than one then select randomly a pair of variables in two lists of domain and range variables such that this pair of variables is different from pairs of variables of role atoms in $roles_p^{body}$ that are SubProperty or SuperProperty of R (the lines 11-12 in Algorithm 34).

– **The length of p is greater than 1 and the head is a role atom (Algorithm 35)**

This function (Algorithm 35) returns the role atom R whose variables is received after removing the unsuitable variables as follows:

* The domain and range variable in the head will be removed from two lists of domain and range variables respectively if R is SubProperty or SuperProperty of the head (the lines 3-5 in Algorithm 35).
* The variables of the concept atoms in the body will be removed from the list of variables of domain or range if these concept atoms subsume R.Domain or R.Range respectively (the lines 7-10 in Algorithm 35).
* If the pattern p does not contains the fresh variable then the value of the output R will be received from the function NoFreshVariablesExist() in Algorithm 33; otherwise if the pattern p contains the fresh variable then

Algorithm 35: The function AdjustRoleAtomWithPHeadIsRoleAtom(): Adjust the variables in the role atom to ensure the language bias is respected with p.length()>1 and p.headAtom is a role atom

input : p: the input pattern; R: the role atom needs to be adjusted variables;
$vars_p$: variable names appearing in the pattern p;

output: R: The role atom have been adjusted variables based on p

$vars_{domain} \leftarrow vars_p$;
$vars_{range} \leftarrow vars_p$;
if *R.isSubPropertyOf(p.headAtom) or R.isSuperPropertyOf(p.headAtom)* **then**
    $vars_{domain} \leftarrow vars_{domain} \setminus p.headAtom.Domain.Var$;
    $vars_{range} \leftarrow vars_{range} \setminus p.headAtom.Range.Var$;
end
$subDomainR_p = concepts_p^{body}.getConceptsSubsuming(R.Domain)$;
$subRangeR_p = concepts_p^{body}.getConceptsSubsuming(R.Range)$;
$vars_{domain} \leftarrow vars_{domain} \setminus subDomainR_p.Vars$;
$vars_{range} \leftarrow vars_{range} \setminus subRangeR_p.Vars$;
if $\neg p.isExistFreshVariable()$ **then**
    $R_{temp} \leftarrow NoFreshVariablesExist()$ in Alg 33;
end
else
    $R_{temp} \leftarrow FreshVariablesExist()$ in Alg 34;
end
$R.Domain.Var \leftarrow R_{temp}.Domain.Var$;
$R.Range.Var \leftarrow R_{temp}.Range.Var$;
return R

the value of the output R will be received from the function FreshVariablesExist() in Algorithm 34 (the lines 11-15 in Algorithm 35).

A.1.3 Pattern adjustment

Algorithm 36: The function AdjustPattern(): Adjust the pattern to become the closed rule

input : p: the input pattern;
$vars_p$: variable names appearing in the pattern p;

output: p: The pattern have been adjusted to become the closed rule

```
1  z ← GetFreshVariable();
2  vars_p ← vars_p \ z;
3  R ← the role atom contains GetFreshVariable();
4  if z is the domain variable then
5  |  Select randomly a variable var_p ∈ vars_p such that var_p is different from
   |   variables of concept atoms in concepts_p^body that subsumes the domain of
   |   R;
6  end
7  else if z is the range variable then
8  |  Select randomly a variable var_p ∈ vars_p such that var_p is different from
   |   variables of concept atoms in concepts_p^body that subsumes the range of R;
9  end
10 Substitute var_p for z;
11 return p
```

This function is used to adjust the rule corresponding to the pattern p to become a closed rule if that rule is a open rule with one fresh variable. This function mainly support for the generalization operator (Section 5.3.2.6) in the evolutionary algorithm.

Bibliography

[+04] Horrocks , Ian , Patel-Schneider , Peter F, Boley , Harold , Tabet , Said , Grossof , Benjamin , Mike Dean, and Mike . Swrl: A semantic web rule language combining owl and ruleml. 21, 01 2004. (Cited in page 39.)

[AAC] Paul Buitelaar Anja Jentzsch Andrejs Abele, John P. McCrae and Richard Cyganiak. http://lod-cloud.net/. (Cited in pages v and 2.)

[AIS93] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993. (Cited in page 33.)

[AJ07] Paulo J. Azevedo and Alípio M. Jorge. Comparing rule measures for predictive association rules. In Joost N. Kok, Jacek Koronacki, Raomon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron, editors, *Machine Learning: ECML 2007*, pages 510–517, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. (Cited in page 15.)

[BA99] Roberto J. Bayardo, Jr. and Rakesh Agrawal. Mining the most interesting rules. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 145–154, New York, NY, USA, 1999. ACM. (Cited in page 15.)

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. (Cited in page 19.)

[BFOS84] L. Breiman, J. Friedman, R. Olshen, and C. Stone. Classification and regression trees. NewYork, USA, 1984. (Cited in page 37.)

[BMUT97] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur. Dynamic itemset counting and implication rules for market basket data. *SIGMOD Rec.*, 26(2):255–264, June 1997. (Cited in page 36.)

[CB91] Peter Clark and Robin Boswell. Rule induction with cn2: Some recent improvements. pages 151–163. Springer-Verlag, 1991. (Cited in page 36.)

[DeJ02] K. DeJong. *Evolutionary Computation: A unified approach.* MIT Press, Cambridge, MA, 2002. (Cited in page 66.)

[DFZD01] Alexandre Delteil, Catherine Faron-Zucker, and Rose Dieng. Learning Ontologies from RDF annotations. In Alexander Maedche, Steffen Staab, Claire Nedellec, and Eduard H. Hovy, editors, *IJCAI 2001 Workshop on Ontology Learning, Proceedings of the Second Workshop on Ontology Learning OL 2001, Seattle, USA, August 4, 2001 (Held in conjunction with the 17th International Conference on Artificial Intelligence IJCAI 2001)*, volume 38 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2001. (Cited in page 12.)

[Div06] Federico Divina. Evolutionary concept learning in first order logic: An overview. *AI Commun.*, 19(1):13–33, January 2006. (Cited in page 14.)

[DM02] Federico Divina and Elena Marchiori. Evolutionary concept learning. In *in GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 9–13. Morgan Kaufmann Publishers, 2002. (Cited in page 14.)

[Dž96] Sašo Džeroski. Advances in knowledge discovery and data mining. chapter Inductive Logic Programming and Knowledge Discovery in Databases, pages 117–152. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1996. (Cited in page 19.)

[ES03] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. 2003. (Cited in page 66.)

[FdE08a] N. Fanizzi, C. d'Amato, and F. Esposito. Learning with kernels in description logics. In F. Zelezný and N. Lavrac, editors, *ILP 2008*, pages 210–225. Springer, 2008. (Cited in page 37.)

[FdE08b] Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. Dl-foil concept learning in description logics. In Filip Železný and Nada Lavrač, editors, *Inductive Logic Programming*, pages 107–121, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. (Cited in page 13.)

[FS00] L. M. Fu and E. H. Shortliffe. The application of certainty factors to neural computing for rule discovery. In *IEEE TRANS. On Neural Networks*, pages 647–657, 2000. (Cited in page 36.)

[GEP04] Gunnar AAstrand Grimnes, Pete Edwards, and Alun Preece. Learning meta-descriptions of the foaf network. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *The Semantic Web – ISWC 2004*, pages 152–165, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. (Cited in page 12.)

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. 1989. (Cited in page 66.)

[GTHS13] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: Association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 413–422, New York, NY, USA, 2013. ACM. (Cited in pages 13, 15, 30, 33, 35, 58, 59, 83, 91 and 129.)

[HKS06] Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible sroiq. In *Proceedings of the Tenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'06, pages 57–67. AAAI Press, 2006. (Cited in page 22.)

[HLA08] Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of owl class descriptions on very large knowledge bases. In *Proceedings of the 2007 International Conference on Posters and Demonstrations - Volume 401*, ISWC-PD'08, pages 102–103, Aachen, Germany, Germany, 2008. CEUR-WS.org. (Cited in page 13.)

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975. (Cited in page 66.)

[HPS04] Ian Horrocks and Peter F. Patel-Schneider. A proposal for an owl rules language. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 723–731, New York, NY, USA, 2004. ACM. (Cited in page 30.)

[HPSB+04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A semantic web rule language combining OWL and RuleML. W3c member submission, World Wide Web Consortium, 2004. Accedido: 21/05/2015. (Cited in page 28.)

[JLL10a] Joanna Józefowska, Agnieszka Lawrynowicz, and Tomasz Lukaszewski. The role of semantics in mining frequent patterns from knowledge bases in description logics with rules. *TPLP*, 10(3):251–289, 2010. (Cited in page 13.)

[JLL10b] Joanna Józefowska, Agnieszka Lawrynowicz, and Tomasz Lukaszewski. The role of semantics in mining frequent patterns from knowledge bases in description logics with rules. *CoRR*, abs/1003.2700, 2010. (Cited in pages 30 and 54.)

[KC04] Graham Klyne and Jeremy J. Carroll. Resource description framework (rdf): Concepts and abstract syntax. W3C Recommendation, 2004. (Cited in page 1.)

[Leh06] Jens Lehmann. Concept learning in description logics. Master's thesis, TU Dresden, 2006. Diploma Thesis in Computer Science, supervisors: Dr. habil. Pascal Hitzler, Prof. Steffen Hölldobler. (Cited in page 14.)

[Leh09] Jens Lehmann. Dl-learner: Learning concepts in description logics. *J. Mach. Learn. Res.*, 10:2639–2642, December 2009. (Cited in page 13.)

[LH07] Jens Lehmann and Pascal Hitzler. Foundations of refinement operators for description logics. In *Inductive Logic Programming, 17th International Conference, ILP 2007, Corvallis, OR, USA, June 19-21, 2007, Revised Selected Papers*, pages 161–174, 2007. (Cited in page 41.)

[MDRP+12] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue, and Ashwin Srinivasan. Ilp turns 20. *Machine Learning*, 86(1):3–23, Jan 2012. (Cited in page 19.)

[MR94] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994. (Cited in page 19.)

[MS] Alexander Maedche and Steffen Staab. Ontology learning. In *HANDBOOK ON ONTOLOGIES*, pages 173–189. Springer. (Cited in page 12.)

[MSS05] Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for owl-dl with rules. *Web Semant.*, 3(1):41–60, July 2005. (Cited in page 30.)

[Mug91] Stephen Muggleton. Inductive logic programming. *New Gen. Comput.*, 8(4):295–318, February 1991. (Cited in page 19.)

[Mun10] D Mun. Knowledge-based part similarity measurement utilizing ontology and multi-criteria decision making technique. *Advanced Engineering Informatics*, 2010. (Cited in page 28.)

[MZ02] Alexander Maedche and Valentin Zacharias. Clustering ontology-based metadata in the semantic web. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Principles of Data Mining and Knowledge Discovery*, pages 348–360, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. (Cited in page 12.)

[NB12] Victoria Nebot and Rafael Berlanga. Finding association rules in semantic web data. *Know.-Based Syst.*, 25(1):51–62, February 2012. (Cited in page 12.)

[NCW97] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. *Foundations of Inductive Logic Programming*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997. (Cited in page 41.)

[Obi] Marek Obitko. https://www.obitko.com/tutorials/ontologies-semantic-web/owl-dl-semantics.html. (Cited in pages v, 26 and 27.)

[QKQ13] ohamad Saraee Qudamah K. Quboa. A state-of-the-art survey on semantic web mining. *Intelligent Information Management*, 5(1), 2013. (Cited in page 11.)

[SG91] P. Smyth and R.M. Goodman. *Rule induction using information theory*. MIT Press, Cambridge, MA, USA, 1991. (Cited in page 37.)

[SHB06] Gerd Stumme, Andreas Hotho, and Bettina Berendt. Semantic web mining. *Web Semant.*, 4(2):124–143, June 2006. (Cited in page 11.)

[SM99] S. Sahar and Y. Mansour. An empirical evaluation of objective interestingness criteria. In *SPIE Conference on Data mining and Knowledge Discovery*, pages 63–74, 1999. (Cited in page 36.)

[SP07] Evren Sirin and Bijan Parsia. Sparql-dl: Sparql query for owl-dl. In *In 3rd OWL Experiences and Directions Workshop (OWLED-2007*, 2007. (Cited in pages 100 and 101.)

[SPG+07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *Web Semant.*, 5(2):51–53, June 2007. (Cited in page 80.)

[TFZG14] Andrea G. B. Tettamanzi, Catherine Faron-Zucker, and Fabien Gandon. Testing owl axioms against rdf facts: A possibilistic approach. In Krzysztof Janowicz, Stefan Schlobach, Patrick Lambrix, and Eero Hyvönen, editors, *Knowledge Engineering and Knowledge Management*, pages 519–530, Cham, 2014. Springer International Publishing. (Cited in page 15.)

[TKS04] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right objective measure for association analysis. *Inf. Syst.*, 29(4):293–313, June 2004. (Cited in page 14.)

[T.R16] B.L.Shivakumar T.Raji. A survey on semantic web mining technologies and web mining tools. *International Journal of Advanced Computational Engineering and Networking*, 4, October 2016. (Cited in page 11.)

[VN11] Johanna Völker and Mathias Niepert. Statistical schema induction. In *Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web: Research and Applications - Volume Part I*, ESWC'11, pages 124–138, Berlin, Heidelberg, 2011. Springer-Verlag. (Cited in page 13.)

[W3Ca] W3C. https://www.w3.org/tr/2012/note-owl2-manchester-syntax-20121211/. (Cited in page 28.)

[W3Cb] W3C. https://www.w3.org/tr/owl2-overview/. (Cited in page 25.)

[W3Cc] W3C. https://www.w3.org/tr/owl2-profiles/. (Cited in page 28.)