Technische Universität Dresden

European Masters Program in Computational Logic

Institute of Theoretical Computer Science

Department of Computer Science

Master Thesis

Specializing conjunctive queries in the $\mathcal{EL}$-family for better comprehension of result sets.

Author
Premchand Nutakki

Advisor
Dr. Ing. Anni-Yasmin Turhan

Overseeing Professor
Prof. Dr.-Ing. Franz Baader

October, 2011

Technische Universität Dresden

Author:	: **Premchand Nutakki**
Matrikel-Nr.	: **3730656**
Title	: **Specializing conjunctive queries in the $\mathcal{EL}$-family for better comprehension of result sets.**
Degree	: **Master of Science**
Major	: **Computational Logic**
Date of Defense	: **October 14, 2011**

Declaration

Hereby I certify that the thesis has been written by me. Any help that I have received in my research work has been acknowledged. Additionally, I certify that I have not used any auxiliary sources and literature except those cited in the thesis.

Premchand Nutakki

Abstract

A recurring problem in many ontology based applications is delivering a huge result set for user's query causing an *information overload*. The reasons for this could be in-accurateness of the query provided by the user, a naive user who is not familiar with ontologies, or existence of a huge knowledge base. Though various techniques to reduce information overload are developed for web based search and Database querying, Description Logic knowledge bases are in need of customized techniques to solve this problem.

In this thesis, we present two approaches to solve the problem of information overload of conjunctive query result sets over $\mathcal{EL}^{++}$-KBs. The first approach is based on the minimal query intensification such that the new queries (called *specializations*) obtained are syntactically similar to the original query. The second approach is based on clustering of query results. We also implemented our two approaches in Java on top of Pellet reasoner and tested them for their effectiveness using an extended LUBM benchmark ontology.

Acknowledgements

First of all I would like to thank my advisor Dr. Ing. Anni-Yasmin Turhan for her abundant support, advice and patience during the completion of this thesis. I would also like to thank all the staff members who guided me during my studies at TU Dresden and TU Wien.

I am indebted to the European Commission for the Erasmus Mundus Scholarship I received without which this journey would never be possible.

Last but not the least, I would like to thank my family and friends for all their invaluable support.

Contents

List of Figures **vi**

List of Tables **vii**

1 Motivation **1**
1.1 Introduction 2
1.2 Related Work 4

2 Preliminaries **7**
2.1 Description Logics 7
2.2 $\mathcal{EL}$ and $\mathcal{EL}$ family 8
2.3 Conjunctive Queries 10

3 Minimal Syntactic Query Intensification **12**
3.1 Specialization and Minimal Specialization 12
3.2 Computing the minimal specializations 17
3.3 Algorithm: *MinSpec* 38

4 Clustering of Query Results **42**
4.1 Clustering 43
4.2 Specializing a query using clustering 46
4.3 Generating clusters and c-specializations 49
4.4 Algorithm: CLUSTERR 58

5 Implementation and Evaluation **60**
5.1 Implementation 60
5.2 Evaluation Test Data 62
5.3 Evaluation on Quality and Performance 63

Conclusion **70**

Bibliography **73**

List of Figures

4.1 Partitional Clustering 43
4.2 Hierarchical Clustering 44
4.3 Partitional Clustering from a Hierarchical Clustering 44
4.4 Four key steps of Feature Selection 45
4.5 Generation of a cluster tree 49
4.6 Sample tree of a single type: Student 54
4.7 Generation of a single Tree 55
4.8 Product Operator 56
4.9 A Tree obtained by using the cross product operator $\otimes$. 57
4.10 Algorithm CLUSTERR 59

5.1 Main view of the application to specialize a query 61
5.2 Effectiveness measure algorithm 65
5.3 Effectiveness measure of the 4 queries using the 3 approaches . . 66
5.4 Increase in effectiveness after each iteration for the two approaches 68

List of Tables

5.1 The list of queries used in the evaluation 63
5.2 Result tuple subsets used for each query 64
5.3 Computation time for *MinSpec* and CLUSTERR 69

Chapter 1

Motivation

With the development and potential usage capabilities of the Semantic Web and Semantic Technologies, ontologies are becoming one of the most prominent paradigms for knowledge representation and reasoning. "Description Logics are knowledge representation formalisms that provide, for example, the logical underpinning of the W3C OWL[1] standards" [Glimm and Rudolph, 2010]. The rise in the popularity of Description Logics (DLs) and semantic web is mainly due to their ability to perform reasoning and at the same time provide results to a query in reasonable time. However in few cases, the set of results obtained by the user's query to the DL knowledge bases are mostly unintended (and irrelevant to the user). Due to this, the user has to analyse the results to obtain the intended information which is a tedious task. This problem of extracting the indented data amongst the results is called *information overload*. Though in web documents based querying and Database querying considerable research is done on how to reduce information overload, there isn't any significant work done for DL Knowledge bases querying. There exists numerous DLs and providing a common solution is not intended here. So, before we look into this problem more deeply we first restrict the DLs that we investigate.

Among the DLs, the $\mathcal{EL}$ family [Baader *et al.*, 2005] has a nice combination of reasonable-expressiveness and acceptable complexity. Real-world ontologies like The Gene Ontology[2], NCI[3] and SNOMED CT[4] [Corent and Keizer, 2008] can be expressed with-in $\mathcal{EL}^{++}$. OWL 2 EL[5] profile is a subset of OWL 2 DL[6] and is based on DL $\mathcal{EL}^{++}$and it aims at applications that employ large ontologies. This profile is sufficiently expressive for many biomedical ontologies like GALEN [Rector and Horrocks, 1997] also. So, from now we restrict ourselves to

[1] http://www.w3.org/2004/OWL/
[2] http://www.geneontology.org/
[3] http://ncit.nci.nih.gov/
[4] http://www.ihtsdo.org/snomed-ct/
[5] http://www.w3.org/TR/owl2-profiles/
[6] http://www.w3.org/TR/owl2-overview/

the knowledge bases in the $\mathcal{EL}$ family. Conjunctive queries (CQs), which is the most common query language used in databases, is also being considered as the most suitable language for querying DL KBs as CQs can sufficiently express the queries over DL KBs. Moreover, answering conjunctive queries has been studied extensively and its results corresponding to various DLs are available [Rosati, 2007][Kroetzsch and Rudolph, 2007]. In this thesis, a query always refers to a conjunctive query.

To reduce the information overload of query results, we need to withhold the unintended data and present a subset of the query results containing the relevant data. This can be achieved by specializing the query which is the aim of our thesis i.e.

To reduce the information overload by specializing the conjunctive query q w.r.t. an $\mathcal{EL}^{++}$-knowledge base $\mathcal{K}$.

and is focused on the naive users who

1. don't have a good understanding of the knowledge base (ontology)
2. are not familiar with description logics
3. are interested in the reduction of the query result set.

1.1 Introduction

As in DL KB, database query results (or result set) and web document search results also face the problem of information overload. Considerable research has been done in database querying and web document search to avoid information overload. The three primary techniques devised to reduce information overload involve use of ranking, categorization, or clustering. *Ranking* returns the results as a list which are ordered based on some metrics, where the most relevant results as placed first and the least relevant results are placed last. For web document searches, tools like Google[7], Bing[8] perform ranking of the search results and are constantly used. A specialized result set is obtained by just selecting the first k results from the ranked list [Agarwal *et al.*, 2003].

Categorization assigns results to pre-defined categories which allows the user to select the relevant categories only. A word *Cricket* can refer either to an insect, or to a sport. So, results related to *Cricket* can be placed in the pre-defined categories *Insect* or *Sport*. *Clustering* groups items based on some similarity measure which often depends on the contents of the clusters. A cluster is identified using labels which helps in the selection of the relevant clusters. Engines like Carrot2[9] and Clusty[10] are developed for clustering web search results.

[7] www.google.com
[8] www.bing.com
[9] http://search.carrot2.org/stable/search
[10] www.clusty.com

[Daniels, 2006] clusters database query results.

However, research on how to reduce the information overload of a query result set over a DL KB is still in an early phase. In this thesis, we propose two approaches to solve our problem, (1) a syntactic approach based on (minimal) query intensification [Bosc *et al.*, 2008], (2) an approach based on clustering of query results. The two approaches differ on the following issues, (a) level of specialization, (b) the usage of the result tuples of the query, and (c) user-dependency.

The syntactic approach computes queries (which yield a specialized results) by intensifying the original query using the knowledge base. These queries, called *specializations*, are also syntactically similar to the original query. Due to the existence of a TBox and RBox, this intensification step can be performed conveniently and systematically. Since the approach is unsupervised, the new queries generated may omit some of the relevant results. For this reason only minimal intensifications are computed, and presenting the user with the set of all minimal specializations gives him/her the sufficient choices to generate the query which reduces (or even avoids) the information overload by a significant amount.

The cluster based approach clusters the query result tuples into n clusters (of approximately the same size), where n is a number provided by the user. To provide better understandability to the user, we also generate n queries, called *c-specializations* (whose results form the clusters), corresponding to each cluster using their labels. Among the various types of clustering structures, and extended form of divisive hierarchical clustering [Manning *et al.*, 2008] is a clear choice as it can be computed easily using a DL knowledge base. The result variables of the query are used to obtain the required features for clustering. The stopping criteria for the divisive hierarchical clustering is based on the number of clusters requested by the user.

In Section 1.2, we present the work done on clustering of query results, and introduce the DLs (related to $\mathcal{EL}$-family) to which CQ answering is decidable. Since, CQ answering is not decidable in full $\mathcal{EL}^{++}$, we need to use a fragment of $\mathcal{EL}^{++}$ which is expressive enough for well-known ontologies and at the same time decidable w.r.t. CQ answering. In Chapter 2, we present some basic concepts and notions related to description logics (especially $\mathcal{EL}$ family) and conjunctive queries. In Chapter 3, we present the syntactic approach for minimal query intensification. In Section 3.1, we identify and define the interesting query intensifications as *specializations* of the query and present the various ways to obtain these specializations w.r.t. a $\mathcal{EL}^{++}$-KB. We also define *minimal specializations* and argue why it is sufficient to compute only the minimal specializations of a query to solve our problem. In section 3.2, we present methods to compute minimal specializations w.r.t. each kind of specialization and in Section 3.3, we present the complete algorithm to compute all the minimal specializations of a query and prove the completeness and soundness of our method. We also

investigate the complexity of our method.

In Chapter 4, we introduce the clustering based approach and argue how we can overcome difficulties of the previous approach. In Section 4.1, we introduce clustering and some of the sub-tasks in a clustering algorithm like feature selection, optimal cluster count. In section 4.2, we show how a query can be specialized by clustering the query results. In Section 4.3, we explain the main steps of our computation method along with the optimizations steps. In Section 4.4, we present the clustering based algorithm to reduce information overload of query results over a DL KB. In Chapter 5, we present the implementation details and optimization steps for the two approaches. We also present the experimentation set-up, the benchmark ontology, sample queries and the results of the performance evaluation. We evaluate the performance using an effectiveness measure for the test cases where some of the result tuples are selected as relevant (or irrelevant) tuples for the user. Finally, we compare the computation time of each approach to check their scalability with huge KBs.

1.2 Related Work

To the best of our knowledge this is the first attempt to devise methods to reduce the information overload by specializing a conjunctive query over a DL knowledge base. One way to accomplish this is to directly alter the query minimally without looking into each result tuple and procure a syntactically similar query which generates a specialized query result set. Clustering of query results is another technique that can be used to specialize a query.

Clustering of query results: The problem of reducing information overload for web document queries using clustering of query results has been studied in great detail. [Calado *et al.*, 2003] proposes a method which automatically predicts the categories using Web link structure. [Kang and Kim, 2003] classifies web document retrieval queries and solves the problem of information overload caused by small queries. [Hammouda and Kamel, 2004] presented two main parts to successful document clustering. However, these methods can be used only for web document search and will be of little use for queries over DL KBs. The reason is that clustering techniques for web document search results concentrates on documents containing huge amounts of unstructured texts while clustering techniques for query results over DL KBs can only deal with tuples consisting of small amount of data. As in databases, the query results over DL KB are also a set of result tuples. Nonetheless, some principles like cluster structures, cluster labelling, feature selection, calculating the number of clusters, selecting the cluster to split will be transferred.

For clustering of query results over Databases, [Chu *et al.*, 1996] proposed an error-based conceptual clustering method for identifying high level concepts of numerical attribute values in DBs. [Chakrabarti *et al.*, 2004] proposed a

method to automatically categorize SQL query results and incrementally construct a labelled, hierarchical category structure. [Fu *et al.*, 2004] introduces a hybrid query similarity measure that uses results returned to queries along with query terms to perform clustering. [Daniels, 2006] shows that users' information overload can be minimized by clustering DB query results and a agglomerative (bottom-up) hierarchical clustering algorithm is used to cluster the query results. However, none of these techniques developed for Databases can be directly used for DLs, as in a DL KB there are descriptions with clear semantics (concept and role inclusion axioms) that can be used for clustering. This makes the techniques for Databases and web documents less efficient for DLs and developing better approaches suitable for DLs is certainly possible. Recently, the clustering of query results in Semantic Web or Ontology (or reasoning-) based systems is being studied.

In [Lawrynowicz, 2009] a framework for clustering results of queries over Semantic Web data is proposed. [Lawrynowicz, 2009a] presented an approach that extends SPARQL with clustering abilities by introducing a new statement, CLUSTER BY, into the SPARQL [Prud'hommeaux and Seaborne, 2008] grammar and it also proposes semantics for such an extension. [Amato *et al.*, 2010] proposes a novel approach which overcomes the shortcomings and drawbacks of syntactic approaches. Specifically, answers are grouped taking into account the concept (and role) inclusions available in the underlying knowledge base. We use a similar grouping technique in our clustering based approach to specialize a query and we also address various shortcomings of [Amato *et al.*, 2010] like unavailability of concept and role refinement operators, non-utilization of role inclusions derived from the KB, and so on.

Conjunctive Query Answering: $\mathcal{EL}^{++}$ is a DL expressive enough for various ontologies like SNOMED CT[11], GO [Gene Ontology, 2010], and at the same time has PTIME complexity for many reasoning tasks. But, conjunctive query (CQ) answering in unrestricted $\mathcal{EL}^{++}$ is undecidable and as illustrated in [Kroetzsch and Rudolph, 2007], the combination of role atoms in queries and complex role inclusion axioms can indeed make reasoning significantly more difficult. So, we look into the fragments of $\mathcal{EL}^{++}$ and its related DLs for which CQ answering is decidable.

It is well-known that CQ answering in $\mathcal{EL}$, $\mathcal{ELH}$ and $\mathcal{SHIQ}$ is decidable [Glimm *et al.*, 2007][Rosati, 2007], but is it also decidable for $\mathcal{SHION}$, $\mathcal{SHOIQ}$? [Glimm and Rudolph, 2010] shows that it holds for $\mathcal{SHION}$ and for $\mathcal{SHOIQ}$ knowledge bases also. In addition, they also argued the same for $\mathcal{SROIQ}$ under similar conditions. [Kroetzsch and Rudolph, 2007] introduced a novel algorithm for answering CQs in $\mathcal{EL}^{++}$-knowledge bases, which is worst-case optimal under various assumptions and it also identifies the $\mathcal{EL}^{++}$-fragment of $\mathcal{SROIQ}$ as a decidable sub-DL: *regular* $\mathcal{EL}^{++}$. The authors shows that CQ answering in

[11] http://www.ihtsdo.org/

regular $\mathcal{EL}^{++}$ is decidable and is NP-hard w.r.t. data size or size of the KB. A regular KB contains a partial-ordering among the role names occurring in the KB and once we obtain this partial-ordering, we will obtain a decidable algorithm for CQ answering. Therefore, for the fragments regular $\mathcal{EL}^{+}$, regular $\mathcal{EL}^{++}$ CQ answering is decidable and while presenting our two approaches we assume a regular $\mathcal{EL}^{++}$-KB.

Chapter 2

Preliminaries

2.1 Description Logics

DLs are a group of knowledge representation formalisms that model a given domain in terms of *individuals* (constants), *concepts* (unary predicates), and *roles* (binary predicates) [Baader and Nutt, 2003]. A typical DL Knowledge Base (KB) contains an assertional component $\mathcal{A}$ called the *ABox*, a terminological component $\mathcal{T}$ called the *TBox*, and an RBox $\mathcal{R}$. We can view the TBoxs and the RBoxs as conceptual schemas of DBs and ABoxs as (partial) instantiations of the schemas. As shown in *Example 1*, a TBox $\mathcal{T}$ consists of *general concept inclusions* which relate two or more concepts, a RBox $\mathcal{R}$ consists of *role inclusions* which relate two or more simple roles and an ABox consists of *assertions* which either relate a concept and an individual or a role and a pair of individuals. The syntax of DLs can be restricted in a variety of ways to trade-off the expressive power against computational complexity, and thus obtain a representation language that is suitable for the application at hand.

Example 1. *A knowledge base* $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ *is*

$\mathcal{T}$:

$\{\top \sqsubseteq mortal \sqcup deity \sqcup hero$

$\forall hasParent.mortal \sqsubseteq mortal$

$hero \sqsubseteq \exists hasAncestor.deity$

$deity \sqsubseteq \forall hasAncestor.deity\ \}$

$\mathcal{R}$:

$\{\ hasParent \sqsubseteq hasAncestor$

$hasAncestor \circ hasAncestor \sqsubseteq hasAncestor\ \}$

$\mathcal{A}$:

$\{$ *hasParent(Hercules,Zeus)*

hasParent(Hercules,Alcmena)

hasParent(Perseus,Zeus)

deity(Zeus)

hero(Perseus)

hasParent(Alcmena,Electryon)$\}$

The main advantage of DL systems is their ability to perform reasoning. Many applications require reasoning for the KBs and by using an appropriate DL with sufficient expressibility we can perform the reasoning quite efficiently. We use the letters $A, B, C, D, E, \ldots$ to present concepts and the letters $r, s, r_1, r_2, \ldots$ to represent roles. The $\mathcal{EL}$ family are identified as DLs having interesting expressibility and which allow tractable reasoning.

2.2 $\mathcal{EL}$ and $\mathcal{EL}$ family

Let N_I, N_C, and N_R be countably infinite and disjoint sets of *individual names*, *concept names*, and *role names* respectively. *$\mathcal{EL}$-concepts* C are built according to the syntax rule $C := \top \mid A \mid C \sqcap D \mid \exists r.C$, where A ranges over N_C, r ranges over N_R, and C, D range over $\mathcal{EL}$-concepts. $\mathcal{ELH}$ extends $\mathcal{EL}$ by allowing a RBox with *simple role inclusions (SRIs)* of the form $\mathrm{r}_1 \sqsubseteq r_2$, where r_1 and r_2 are role names. $\mathcal{EL}^+$ extends $\mathcal{EL}$ by also allowing in the RBox *role inclusions (RIs)* of the form $r_1 \circ \cdots \circ r_n \sqsubseteq r_{n+1}$, where each r_i is a role name. Transitive roles are expressed by $r \circ r \sqsubseteq r$. Finally, $\mathcal{EL}^{++}$ extends $\mathcal{EL}^+$ by allowing the new concept expressions $\bot$, nominals $\{a\}$ and the concrete domain constructor $p(f_1, f_2, \ldots, f_n)$. The concrete domain constructor is used to refer strings and integers. In this thesis, we assume that all the knowledge bases and CQs don't contain concrete domains.

The semantics of $\mathcal{EL}^{++}$-concepts and roles are defined using an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where the interpretation *domain* $\Delta^{\mathcal{I}}$ is a non-empty set, and $\cdot^{\mathcal{I}}$ is a function mapping each concept name A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$ and each role name $r^{\mathcal{I}}$ to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The function $\cdot^{\mathcal{I}}$ is inductively extended to arbitrary concepts by setting $\top^{\mathcal{I}} := \Delta^{\mathcal{I}}$, $\bot^{\mathcal{I}} := \emptyset$, $\{a\}^{\mathcal{I}} := a^{\mathcal{I}}$, $(C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}$, and $(\exists r.C)^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid \exists e \in C^{\mathcal{I}} : (d, e) \in r^{\mathcal{I}}\}$. The function $\cdot^{\mathcal{I}}$ is inductively extended to arbitrary roles by setting $(r_1 \circ r_2)^{\mathcal{I}} := \{(x, y) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid \exists z \in \Delta^{\mathcal{I}} \wedge (x, z) \in r_1^{\mathcal{I}} \wedge (z, y) \in r_2^{\mathcal{I}}\}$.

An $\mathcal{EL}^{++}$-KB consists of an ABox $\mathcal{A}$, a TBox $\mathcal{T}$ and a RBox $\mathcal{R}$. Typically an ABox $\mathcal{A}$ consists of assertions of the form $C(a)$ or $r(a, b)$. In *Example 2*, $Person(Adam)$ means *Adam* is a $Person$ and $hasChild(Alcmena, Hercules)$ specifies that *Alcmena* has a child named $Hercules$. An interpretation $\mathcal{I}$ *satisfies* an assertion of the form $C(a)$, $r(a, b)$ (i.e. $\mathcal{I} \models C(a)$, $\mathcal{I} \models r(a, b)$) if $a \in C^{\mathcal{I}}$, $(a, b) \in r^{\mathcal{I}}$ respectively. $\mathcal{I}$ is a *model* of an ABox $\mathcal{A}$ if it satisfies all the assertions in $\mathcal{A}$. $\mathcal{A} \models C(a)$ if every model of $\mathcal{A}$ satisfies $C(a)$.

Example 2. *ABox assertions in a simple $\mathcal{EL}^{++}$-ABox $\mathcal{A}_1$*

Women(Mary), Person(Adam), Father(Adam)
Mother(Eve), Son(Hercules), hasChild(Alcmena,Hercules)
hasAncestor(Hercules,Adam) hasHusband(Deianeira,Hercules)

A TBox $\mathcal{T}$ is a finite set of *general concept inclusions (GCIs)* $C \sqsubseteq D$, where C and D are concepts. Example 3 specifies a simple $\mathcal{EL}^{++}$-TBox. An interpretation $\mathcal{I}$ *satisfies* a GCI $C \sqsubseteq D$ (written $\mathcal{I} \models C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$ if it satisfies all GCIs in $\mathcal{T}$. Here, D describes necessary conditions for being a C. $A \equiv C$ is an $\mathcal{EL}^{++}$ *concept definition* for $A \in \mathrm{N}_C$ and it expresses the two CGIs $A \sqsubseteq C$ and $C \sqsubseteq A$. C describes the necessary and sufficient conditions for being an A.

Example 3. *Concept inclusions in a simple $\mathcal{EL}^{++}$-TBox $\mathcal{T}_1$*

$\{$ *Father* $\equiv$ *Person* $\sqcap$ *Male* $\sqcap \exists$*hasChild.*$\top$ *Father* $\sqsubseteq \exists$*hasChild.*$\top$

$\mathit{Student} \equiv \mathit{Person} \sqcap \exists \mathit{is\text{-}registered\text{-}at.University}$ $\quad \mathit{Father} \sqsubseteq \mathit{Person}$
$\mathit{Human} \sqsubseteq \exists \mathit{eats}.\top$ $\quad \mathit{Plant} \sqsubseteq \exists \mathit{grows\text{-}in.Area}$ $\mathit{Vegetarian} \sqsubseteq \mathit{Healthy}\}$

A RBox $\mathcal{R}$ consists of *role inclusions* of the form $s \sqsubseteq r$ or $r_1 \circ \cdots \circ r_n \sqsubset r$, where $s, r, r_1, r_2, \cdots, r_n$ are role names. Example 4 specifies a simple $\mathcal{EL}^{++}$-RBox. An interpretation $\mathcal{I}$ *satisfies* a simple role inclusion $r_1 \sqsubseteq r_2$ (written $\mathcal{I} \models r_1 \sqsubseteq r_2$) if $r_1^{\mathcal{I}} \subseteq r_2^{\mathcal{I}}$. $\mathcal{I}$ *satisfies* a role inclusion $r_1 \circ \cdots \circ r_n \sqsubseteq r$ if $r_1^{\mathcal{I}} \circ \cdots \circ r_n^{\mathcal{I}} \subseteq r^{\mathcal{I}}$. $\mathcal{I}$ is a *model* of a KB $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ if $\mathcal{I}$ is a *model* of $\mathcal{T}$, $\mathcal{I}$ is a *model* of $\mathcal{R}$ and $\mathcal{I}$ is a *model* of $\mathcal{A}$.

Example 4. *Role inclusions in a simple $\mathcal{EL}^{++}$-RBox $\mathcal{R}_1$*
$\mathit{hasSon} \sqsubseteq \mathit{hasChild}$ $\quad \mathit{isFather} \sqsubseteq \mathit{isParent}$
$\mathit{hasAncestor} \circ \mathit{hasAncestor} \sqsubseteq \mathit{hasAncestor}$

We write $C \sqsubseteq_{\mathcal{K}} D$ (or $\mathcal{K} \models C \sqsubseteq D$) if every model of $\mathcal{K}$ satisfies $C \sqsubseteq D$ and we say C is *subsumed by* D and D *subsumes* C w.r.t. $\mathcal{K}$. $C \equiv_{\mathcal{K}} D$ specifies that $A \sqsubseteq_{\mathcal{K}} C$ and $C \sqsubseteq_{\mathcal{K}} A$ and we say that C is *equivalent* to D w.r.t. $\mathcal{K}$. Similarly we write $r \sqsubseteq_{\mathcal{K}} s$ (or $\mathcal{K} \models r \sqsubseteq s$) if every model of $\mathcal{K}$ satisfies $r \sqsubseteq s$ and we say that r is subsumed by s and s subsumes r. $r \equiv_{\mathcal{K}} s$ specifies that $r \sqsubseteq_{\mathcal{K}} s$ and $s \sqsubseteq_{\mathcal{K}} r$ and we say that r is *equivalent* to s w.r.t. $\mathcal{K}$.

A standard $\mathcal{EL}^{++}$-TBox $\mathcal{T}$ is in *normal form* [Baader *et al.*,2005] if it consists of concept inclusions of the form:
$A \sqsubseteq B$, where A and B are concept names.
$A_1 \sqcap A_2 \sqsubseteq B$, where A_1, A_2, B are concept names.
$A \sqsubseteq \exists r.B$, where A, B are concept names and r is a role name.
$\exists r.A \sqsubseteq B$, where A, B are concept names and r is a role name.
A standard $\mathcal{EL}^{++}$-RBox $\mathcal{R}$ is in normal form if it consists of role inclusions of the form:
$r_1 \sqsubseteq r$, where r_1, r are role names.
$r_1 \circ r_2 \sqsubseteq r$, where r_1, r_2, r are role names.

A standard KB $\mathcal{K}$ is in normal form if its TBox and RBox are in normal form. Given an $\mathcal{EL}^{++}$-TBox $\mathcal{T}$ and RBox $\mathcal{R}$, one can compute in polynomial time a TBox $\mathcal{T}'$ and RBox $\mathcal{R}'$ in normal form as shown in [Baader *et al.*, 2005]. By introducing new concept and role names, any TBox $\mathcal{T}$ and RBox $\mathcal{R}$ can be turned into a normalized TBox $\mathcal{T}'$ and RBox $\mathcal{R}'$ respectively, that is a conservative extension of $\mathcal{T}$ and $\mathcal{R}$. Every model of $\mathcal{T}'$ (and $\mathcal{R}'$) is also a model of $\mathcal{T}$ (and $\mathcal{R}$), and every model of $\mathcal{T}$ (and $\mathcal{R}$) can be extended to a model of $\mathcal{T}'$ (and $\mathcal{R}'$) by appropriately choosing the interpretations of the additional concept and role names.

Definition 2.1. *An $\mathcal{EL}^{++}$-RBox in normal form is* regular *if there is a strict partial order $\prec$ on N_R such that, for all role inclusion axioms $r_1 \sqsubseteq s$ and $r_1 \circ r_2 \sqsubseteq s$, we find $r_i \prec s$ or $r_i = s$ (i = 1, 2). An $\mathcal{EL}^{++}$ knowledge base is regular if it has a regular RBox.*

Accordingly, we define a *simple* RBox as below.

Definition 2.2. *An RBox is* simple *whenever, for all axioms of the form* $r_1 \circ s \sqsubseteq s$, $s \circ r_2 \sqsubseteq s$, *the RBox does not contain a common subrole* $r^{'}$ *of* r_1 *and* r_2 *for which there is an axiom of the form* $r \circ s^{'} \sqsubseteq r^{'}$ *or* $s^{'} \circ r \sqsubseteq r^{'}$. *An* $\mathcal{EL}^{++}$*-KB is* simple *if it has a simple RBox.*

We consider a KB to be regular when we present a computation method and we consider a RBox to be also simple whenever we talk about complexity. The set of concept names and role names occurring in a KB $\mathcal{K}$ are represented by $\mathrm{N}_C^{\mathcal{K}}$ and $\mathrm{N}_R^{\mathcal{K}}$ respectively.

2.3 Conjunctive Queries

Conjunctive Queries (CQs) are restricted forms of the first-order queries which correspond to the fragment of relational algebra (or SQL) queries, namely plain select-project-join queries, and UCQs are simply the unions of such queries. CQs and UCQs are widely studied in databases [Abiteboul *et al.*, 1995], [Chandra and Merlin, 1977], and due to a good trade-off between expressive power and nice computational properties, they have been adopted as core query languages in several contexts, such as query optimization, data integration, and also in ontology-based data access. Most of the work concerning query answering in DLs refers in fact to CQs and UCQs.

Definition 2.3. *A conjunctive query (CQ) is of the form* $q(\vec{x}) \leftarrow \exists \vec{y}.conj(\vec{x}, \vec{y}, \vec{c})$, *where* $q(\vec{x})$ *is the head of* q, $conj(\vec{x}, \vec{y}, \vec{c})$ *is the body of* q, $\vec{x}$ *are* distinguished variables, $\vec{y}$ *are* non-distinguished variables, $\vec{c}$ *are* individuals, *and* $conj(\vec{x}, \vec{y}, \vec{c})$ *is a conjunction of terms of the form* $C(v)$ *or* $r(v_1, v_2)$ *for* C *a (non-atomic) concept,* r *a (non-atomic) role, and* v, v_1 *and* v_2 *are variables in* $\vec{x}$, $\vec{y}$ *or individuals in* $\vec{c}$. *We call* $C(v)$ *a concept term and* $r(v_1, v_2)$ *a role term.*

A simplified representation of conjunctive queries is $q(\vec{x}) \leftarrow p_1 \wedge p_2 \wedge \cdots \wedge p_n$, where $q(\vec{x})$ is the head, $p_1 \wedge p_2 \wedge \cdots \wedge p_n$ is the body, $p_1, p_2, \ldots, p_n$ are the terms occurring in q. Note that, a term p_i can contain a complex concept or a complex role. Let $p_i = C_1 \sqcap C_2 \sqcap \ldots \sqcap C_n\ (x)$ be a concept term, then we call C_1,C_2,...,C_n the conjuncts of the concept term p_i. Example 5 provides two simple conjunctive queries q and q_1.

Example 5. *CQs for a KB with TBox, RBox from Example 3, 4:*
$q(x) \leftarrow \exists y.Person(x) \wedge hasChild(x, y)$
$q_1(x) \leftarrow Father(x) \wedge Student(x).$
Here, q *returns all the* $Persons$ *who* $hasChild$ *while* q_1 *returns all* $Fathers$ *who are also* $Students$.

A CQ is called a *Boolean conjunctive query* (BCQ) if it has no distinguished variables. A *Union of Conjunctive Queries* (UCQ) is a disjunction of CQs i.e. a query $q(\vec{x}) \leftarrow \exists \vec{v}.(\varphi_1(\vec{x}, \vec{v}, \vec{c}), \ldots, \varphi_n(\vec{x}, \vec{v}, \vec{c}))$, where each $\varphi_i(\vec{x}, \vec{v}, \vec{c})$ is a conjunction of terms. The set of variables occurring in the query q and a term tm is represented by $Var(q)$ and $Var(tm)$ respectively. Let $\mathcal{I}$ be an interpretation, q a conjunctive query and π: $Var(q) \rightarrow \Delta^{\mathcal{I}}$ a total function. We write

- $\mathcal{I} \models^{\pi} C(v)$ if $(\pi(v)) \in C^{\mathcal{I}}$;
- $\mathcal{I} \models^{\pi} r(v, v^{'})$ if $(\pi(v), \pi(v^{'})) \in r^{\mathcal{I}}$;

If $\mathcal{I} \models^{\pi} tm, \forall\ tm \in q$, we write $\mathcal{I} \models^{\pi} q$ and call π a *match* for $\mathcal{I}$ and q. We say that $\mathcal{I}$ *satisfies* q and write $\mathcal{I} \models q$ if there is a match π for $\mathcal{I}$ and q. If $\mathcal{I} \models q$ for all models $\mathcal{I}$ of a KB $\mathcal{K}$, we write $\mathcal{K} \models q$ and say that $\mathcal{K}$ *entails* q. The query entailment problem is defined as follows:

Given a knowledge base $\mathcal{K}$ and a query q, decide whether $\mathcal{K} \models q$.

2.3.1 CQ answering and containment

Let q_1 and q_2 be two n-ary (having n distinguished variables) conjunctive queries having the same tuple of distinguished variables and $\mathcal{K}$ is a regular $\mathcal{EL}^{++}$ KB. Let, $q(\mathcal{K})$ represent the result set (a set containing the result tuples) of q over $\mathcal{K}$.

CQ answering problem: given a CQ q and a KB $\mathcal{K}$, compute q($\mathcal{K}$).

We call a query q an empty query w.r.t. $\mathcal{K}$ if $q(\mathcal{K}) = \emptyset$. A query q_1 is *contained* in q_2 w.r.t. $\mathcal{K}$ if $q_1(\mathcal{K}) \subseteq q_2(\mathcal{K})$ i.e. result set of q_1 over $\mathcal{K}$ is a subset of the result set of q_2 over $\mathcal{K}$. It is represented as $q_1 \subseteq_{\mathcal{K}} q_2$.

CQ containment problem: given two CQs q_1 and q_2 and a KB $\mathcal{K}$, is $q_1 \subseteq_{\mathcal{K}} q_2$?

We use $q_1 \subset_{\mathcal{K}} q_2$ to express that $q_1 \subseteq_{\mathcal{K}} q_2$ and $q_1(\mathcal{K}) \neq q_2(\mathcal{K})$ and we say q_1 is *strictly contained* in q_2 w.r.t. $\mathcal{K}$. The CQ containment in $\mathcal{EL}^{++}$is undecidable as it turns out that the CQ containment problem is exactly the same problem as the CQ answering problem [Rosati, 2007], [Abiteboul *et al.*, 1995]. As we mentioned in Section 1.2, for regular $\mathcal{EL}^{++}$ both problems are still decidable.

Chapter 3

Minimal Syntactic Query Intensification

Description Logics are developed to perform reasoning which gives it the ability to deduce implicitly captured knowledge from the KBs. However, users are frequently overwhelmed by the huge number of results generated and face much trouble in selecting the relevant data (i.e. data intended by the user). It can be due to (1) incomplete understanding of the underlying Ontology, (2) existence of a huge KB, (3) a naive user providing an inaccurate query, and so on. This may cause the users to spend lots of time navigating through the results in order to find the relevant ones. In cases like these, to reduce the *information overload*, we can specialize the original query q w.r.t. the KB to obtain one or more new queries having a reduced result set.

In this chapter, we present a new approach based on query intensification to reduce information overload of conjunctive query results over a regular $\mathcal{EL}^{++}$-KB. In Section 3.1, we define a desirable query (i.e. a semantically similar and non-empty query) obtained from query intensification as a *specialization* and present various ways to obtain a specialization. Among the specializations, the specializations with minimal reduction in their result set are defined as *minimal specializations*. We also argue that, computing only the minimal specialization suffices to considerably reduce information overload. In Section 3.2 we present the main steps of our method to calculate all the minimal specializations of a query w.r.t. a KB. Finally, in Section 3.3 we present the complete method and show the soundness, completeness of our method along with the complexity results.

3.1 Specialization and Minimal Specialization

Intuitively, a desirable query obtained from query intensification *(specialization)* of a conjunctive query q is a conjunctive query

- whose results are a subset of the results of the original query q,
- which is syntactically similar to q.

We are not interested in the queries which are syntactically dissimilar to the original query q as it is important to eliminate random queries which are not at all related to the original query, and also for the user understandability on how the specializations are obtained. Without the second condition, any query which is contained in the original query q can be a specialization of q. To be a specialization condition one is fundamental and the reasons are evident.

Before we define a *specialization* of a query w.r.t. a KB, we introduce the various syntactic elements of a conjunctive query and we explain what it means by a syntactically similar query. The syntactic elements in a conjunctive query are roles, concepts, variables and individuals and these 4 elements can be divided into 3 groups (1) Concepts, (2) Roles, (3) Variable-Individual (the set of variables and individuals) based on the arity of the elements. Roles, Concepts, Variable-Individual can be seen as functions with arity two, one, zero respectively. Variables and individuals belong to the same group, because replacing a variable and an individual in a CQ with each other still generates a conjunctive query. Replacing a concept in a CQ with a role or an individual won't generate a valid conjunctive query.

We say that a query q' is syntactically similar to a query q if q' is obtained by either (a) changing only the syntactic elements of a query i.e. a concept, role, individual/variable can only be replaced by another concept, role, individual/variable respectively, or (b) adding a new role term with the same arguments as any one of the role terms in the query.

Definition 3.1. *A conjunctive query q' in $\mathcal{EL}^{++}$ is called a* specialization *of a conjunctive query $q(\vec{x}) \leftarrow p_1 \wedge p_2 \wedge \cdots \wedge p_m$ in $\mathcal{EL}^{++}$ w.r.t. an $\mathcal{EL}^{++}$ KB $\mathcal{K}$ if*

1. *$q'(\mathcal{K}) \subset q(\mathcal{K})$*
2. *$q'(\mathcal{K}) \neq \emptyset$*
3. *either (a) $q'(\vec{x}) \leftarrow p_1 \wedge p_2 \wedge p_{i-1} \wedge p'_i \wedge p_{i+1} \wedge \cdots \wedge p_m$, where p'_i is obtained by replacing one syntactic element in p_i with another of the same type.*
   or (b) $q'(\vec{x}) \leftarrow p_1 \wedge p_2 \wedge \cdots \wedge p_m \wedge p_{m+1}$, where $\exists i \in [1 \ldots m]$: $p_i = r(u, v)$ and $p_{m+1} = s(u, v)$.
4. *$Var(q') = Var(q)$*

It is represented by $q' \subset_{\mathcal{K}} q$. Here u, v can be variables or individuals.

So, a specialization of a query q is obtained by modifying one of the terms in q or by adding a new role term (while complying with few conditions). Since, queries obtained by adding a concept term can be simulated by queries obtaining by replacing a concept, we can exclude adding a new concept term to the

query. The set of specialization of a CQ q w.r.t. $\mathcal{K}$ is represented by $Spec_{\mathcal{K}}(q)$.

The specializations in $Spec_{\mathcal{K}}(q)$ can belong to various levels depending on how strongly they are specialized as explained in Example 6. To obtain specializations in various levels, it is sufficient to change only one syntactic element in a term occurring in the query or add one new role term satisfying condition 3b in Definition 3.1.

Example 6. *Consider a KB* $\mathcal{K}$= ($\mathcal{T}$,$\mathcal{A}$)
$\mathcal{T}$ = { *Coach* $\sqsubseteq$ *Father, Father* $\sqsubseteq$ *Man , Man* $\sqsubseteq$ *Person* }
$\mathcal{A}$ = { *GrandFather(a), Father(b), Man(c), Person(d),*
livesIn(a, m), *livesIn*(b, n), *livesIn*(c, o), *livesIn*(d, p) },
and a query $q(y) \leftarrow \exists y.Person(x) \wedge livesIn(x, y)$
Specializations of q w.r.t. $\mathcal{K}$*:*
$q_1(y) \leftarrow \exists y.Man(x) \wedge livesIn(x, y)$
$q_2(y) \leftarrow \exists y.Father(x) \wedge livesIn(x, y)$
$q_3(y) \leftarrow \exists y.GrandFather(x) \wedge livesIn(x, y)$
The result sets for each of the above 4 queries are $\{m, n, o, p\}$, $\{m, n, o\}$, $\{m, n\}$, $\{m\}$. *Moreover,* $q_3 \subset_{\mathcal{K}} q_2 \subset_{\mathcal{K}} q_1 \subset_{\mathcal{K}} q$. *Here,* q_1 *belongs to level 1,* q_2 *belongs to level 2 and* q_3 *belongs to level 3 of the specializations of q w.r.t.* $\mathcal{K}$. *Thus, only by changing one syntactic element Person, we can obtain various levels of specialization. Thus we restrict the number of changes to one.*

Specialization of a union of conjunctive queries (UCQs) is obtained by simply specializing at least one of the disjuncts in the union or by removing a disjunct. Checking if a query q_1 is a specialization of a query q is at least as hard as the CQ containment checking problem, due to the first condition in Definition 3.1.

Definition 3.2. *Let* q, q_1 *be two conjunctive queries in* $\mathcal{EL}^{++}$. q_1 *is called a* minimal specialization *of q w.r.t. an* $\mathcal{EL}^{++}$ *KB* $\mathcal{K}$ *if (1)* q_1 *is a specialization of q w.r.t.* $\mathcal{K}$ *(i.e.* $q_1 \subset_{\mathcal{K}} q$*), (2) there is no other specialization* q_2 *in* $\mathcal{EL}^{++}$ *of q w.r.t.* $\mathcal{K}$ *such that* $q_1 \subset_{\mathcal{K}} q_2$ *and* $q_2 \subset_{\mathcal{K}} q$. *It is represented by* $q_1 \subset^{min}_{\mathcal{K}} q$.

A set of all minimal specializations of a conjunctive query q w.r.t. $\mathcal{K}$ is denoted by the set $minSpec_{\mathcal{K}}(q)$. In Example 6, only q_1 is a minimal specialization of q w.r.t. $\mathcal{K}$.

3.1.1 Types of Specializations

We now introduce the 6 different types of specializations that can be obtained for a query q in $\mathcal{EL}^{++}$. They address the various syntactic elements appearing in the query while following all the 4 conditions in Definition 3.1.

concept specialization: It is obtained when a concept C occurring in a concept term $C(v)$ of query q is *specialized* w.r.t. $\mathcal{K}$.

We say that a concept C occurring in a query q is *specialized* w.r.t. $\mathcal{K}$ if the concept C is replaced by a ($\mathcal{EL}^{++}$) concept C' where $C' \sqsubset_{\mathcal{K}} C$ and we call the

concept C', a *specialization* of concept C w.r.t. $\mathcal{K}$. The set of specializations of C w.r.t. $\mathcal{K}$ is defined by the set $cSpec_{\mathcal{K}}(C)$ and the set of concept specializations of q w.r.t. $\mathcal{K}$ is defined by the set $\mathbf{cSpec}_{\mathcal{K}}(q)$.

role specialization: It is obtained when a role r occurring in a role term $\mathrm{r}(v_1, v_2)$ of q is *specialized* w.r.t. $\mathcal{K}$.

Here, we say that a role r occurring in a query q is *specialized* w.r.t. $\mathcal{K}$ if the role r is replaced by a ($\mathcal{EL}^{++}$) role r_1, where $r_1 \sqsubset_{\mathcal{K}} r$. Here, r_1 is called a *specialization* of a role r w.r.t. $\mathcal{K}$. The set of role specializations of a query q w.r.t. $\mathcal{K}$ is denoted by the set $\mathbf{RSpec}_{\mathcal{K}}(q)$ and the set of specializations of a role r w.r.t. $\mathcal{K}$ is defined by the set $RSpec_{\mathcal{K}}(r)$.

conjunct specialization: It is a specialization obtained by adding a new role term to the query q.

The set of conjunct specializations of a query q w.r.t. $\mathcal{K}$ is defined by the set $\mathbf{conjSpec}_{\mathcal{K}}(q)$. There are many kinds of role terms that can be added to the query q. To obtain a specialization of q only a role term of the form $r_1(x, y)$, where $r(x, y)$ is a term in q, $r_1 \not\sqsubseteq_{\mathcal{K}} r$, and $r \not\sqsubseteq_{\mathcal{K}} r_1$ can be added (Condition 3b in Definition 3.1). The restriction $r \not\sqsubseteq_{\mathcal{K}} r_1$ eliminates queries which have an equivalent query amongst the role specializations.

variable specialization: When one occurrence of a variable x in q is replaced by another variable y in q, a *variable specialization* is obtained.

The set of variable specializations of a query q w.r.t. $\mathcal{K}$ is defined by the set $\mathbf{varSpec}_{\mathcal{K}}(q)$. Unlike the previous kinds of specialization, replacing one occurrence of a variable with another variable can even increase the size of the query result set. This makes the problem of computing the variable specializations of a query much more harder (when compared to other types of specializations).

variable individual specialization: Replacing one occurrence of a variable x in q with an individual a in the ABox generates a *(varIndv) variable individual specialization.*

The set of variable individual specializations of a query q w.r.t. $\mathcal{K}$ is defined by the set $\mathbf{varIndvSpec}_{\mathcal{K}}(q)$. Replacing a variable with all the individuals in the ABox isn't efficient. We need to obtain additional information from the KB regarding the individuals' ordering w.r.t. each role. For this purpose, we construct *role successor hierarchies* and a *role predecessor hierarchies* corresponding to each role name r occurring in the KB (see Section 3.2.1).

individual specialization: An *individual specialization* is obtained by replacing an individual a occurring in q with another individual b occurring in the ABox.

The set of individual specializations of a query q w.r.t. $\mathcal{K}$ is defined by the set $\mathbf{indvSpec}_{\mathcal{K}}(q)$. Though replacing an individual with another individual rarely produces a specialization, we can use the role successor and role predecessor hierarchies to check the interesting cases only.

3.1.2 Computing the specializations of a query

All specializations in $Spec_{\mathcal{K}}(q)$ can be computed by incrementally specializing the query q in different ways until a set of queries are generated, from which no new specialization can be generated. Thus, generating all the specializations of a query is conceptually simple but computationally inefficient as the size of the set can increase exponentially. Also, we should be cautious when we specialize a query in an automated way. The specializations generated shouldn't be too specific resulting in the omission of some of the relevant result tuples. Moreover, choosing the appropriate specialization(s) for providing to the user is not an easy task.

3.1.3 Advantages of computing minimal specializations

Compared to the number of specializations, the number of minimal specializations of a query are fairly small. Moreover, they can also reduce the information overload by a sufficient amount. In cases where the user wants stronger specializations, he/she can choose the intersection of two or more minimal specialization. As shown in Example 7, the combined result set of such a stronger specialization would be the tuples occurring in all the selected minimal specializations. Alternatively, if all the minimal specializations of a query are too specific and weaker queries are required, we can take the union of two or more minimal specializations to obtain a query with a larger subset of result tuples as shown in Example 7.

Example 7. *Consider a KB $\mathcal{K}$ having the following inclusions*

$\mathcal{T} = \{$ *GraduateStudent* $\sqsubseteq$ *Student, UnderGraduateStudent* $\sqsubseteq$ *Student, ResearchAssistant* $\sqsubseteq$ *Student, TeachingAssistant* $\sqsubseteq$ *Student* $\}$

$\mathcal{R} = \{$ *owns* $\sqsubseteq$ *livesIn* $\}$

and a query $q(y) \leftarrow \exists y.Student(x) \wedge livesIn(x, y)$

Minimal specializations of $q(y)$ w.r.t. $\mathcal{K}$:

$q_1(y) \leftarrow \exists y.GraduateStudent(x) \wedge livesIn(x, y)$

$q_2(y) \leftarrow \exists y.UnderGraduateStudent(x) \wedge livesIn(x, y)$

$q_3(y) \leftarrow \exists y.ResearchAssistant(x) \wedge livesIn(x, y)$

$q_4(y) \leftarrow \exists y.TeachingAssistant(x) \wedge livesIn(x, y)$

$q_5(y) \leftarrow \exists y.Student(x) \wedge owns(x, y)$

As you can see, the above 5 (minimal) specializations can significantly reduce the size of the result set of $q(y)$. We could even obtain a stronger or weaker specialization by the intersection or union of 2 or more of these specializations. $q_1(\mathcal{K}) \cup q_3(\mathcal{K})$ gives the location of all the GraduateStudents who are also ResearchAssistants. To obtain a stronger specialization, we take the common result tuples of 2 or more specializations. $q_1(\mathcal{K}) \cap q_3(\mathcal{K})$ gives the location of all the GraduateStudents who are working as ResearchAssistants, $q_3(\mathcal{K}) \cap q_5(\mathcal{K})$ gives the all locations which are owned by a ResearchAssistant.

From Example 7 we can see that it is efficient to provide the user only the minimal specializations of a query w.r.t. a KB to efficiently solve our problem

of information overload. Generating all the minimal specializations will provide us more options to reduce information overload. Hence, we develop a method to compute all the minimal specializations and show that these specializations are in fact minimal. We do that by obtaining minimal specializations with respect to each kind of specializations and then eliminating the specializations which are not minimal w.r.t. q. In the end, we also perform post-processing steps to eliminate the additional variables introduced in the intermediate steps.

3.2 Computing the minimal specializations

Before starting the computation of the minimal specializations of query, rewriting the query into a normal form w.r.t. $\mathcal{EL}^{++}$ (along with the normalization of the KB [Baader *et al.*, 2005]), can simplify the computation process considerably. We can also extract implicit knowledge from the KB irrespective of the query as a pre-processing step and utilize it while specializing a query. We first start by presenting a new normal form for conjunctive queries w.r.t. $\mathcal{EL}^{++}$, CQNF (Conjunctive Query Normal Form) and then we specify the pre-processing steps required for our method. In the next sections, we compute the minimal specializations w.r.t. each type of specialization. From these specializations we can easily retrieve the minimal specializations of a query. Before presenting our final algorithm, we also introduce the post-processing steps (like removal of new variables, concept and role names, *et al.*) required for better user understandability (of the specializations).

3.2.1 Normalization and pre-processing steps

As we are *specializing* (-the process of generating specializations-) a CQ with respect to the $\mathcal{EL}^{++}$-KB, it is efficient to convert the CQs to an appropriate normal form to make the computation process simpler. We can omit the nominals occurring in a query by performing few rewriting steps. Suppose a query q is of the form $q(y) \leftarrow C \sqcap \{a\}(x) \wedge r(x, y)$. Then we can rewrite q into an equivalent query $q'(y) \leftarrow C(a) \wedge r(a, y)$. First we remove the variable occurring in the term tm with nominal from the query, by replacing all its occurrence with the nominal value (individual). Then we eliminate the nominal from the term tm. Similarly, we can eliminate all the other nominals occurring in the query.

Even the concepts and roles occurring in the terms of a query can be normalized to compute the specializations easily (i.e. omission of the redundant conjuncts of a concept and elimination of role compositions in the query).

Definition 3.3. *Let $C \equiv C_1 \sqcap \cdots \sqcap C_n$. A concept $D \equiv C_{k_1} \sqcap \cdots \sqcap C_{k_i}$ is called a* sub-concept *of C w.r.t. $\mathcal{K}$ if $C_{k_1}, C_{k_2}, \cdots, C_{k_i}$ are i distinct conjuncts in C and $1 \leq i < n \wedge 1 \leq k_i \leq n$. We call a sub-concept C_N of C, a* normalized concept *of C w.r.t. $\mathcal{K}$ if*

- $C_N \equiv_{\mathcal{K}} C$

- *removing any one of the sub-concepts of C_N results in a non-equivalent concept D (i.e. $C_N \sqsubset_{\mathcal{K}} D$).*

The set of all sub-concepts of C is defined by the set $SUB(C)$. A concept name A do not have any sub-concepts the normalized concept of A is A itself. and a non-atomic concept C can have more than 1 normalized concepts and the set of normalized concepts of C w.r.t. $\mathcal{K}$ is represented by $NORM_{\mathcal{K}}(C)$. Consider, $\mathcal{K} = \{B \sqsubseteq A, C \equiv A \sqcap B\}$, then B is the normalized concept of C w.r.t. $\mathcal{K}$. Here, $SUB(C) = \{A, B\}, NORM_{\mathcal{K}}(C) = \{B\}$. Consider, $\mathcal{K}' = \{B \sqsubseteq A, D \equiv B, C \equiv A \sqcap B \sqcap D, \}$, then both B and D are the normalized concepts of C w.r.t. $\mathcal{K}$. Here, $SUB(C) = \{A, B, D, A \sqcap B, A \sqcap D, B \sqcap D\}, NORM_{\mathcal{K}'}(C) = \{B, D\}$ and $NORM_{\mathcal{K}'}(B) = B$. For any concept C we can easily generate its set of normalized concepts, $NORM_{\mathcal{K}}(C)$.

Now we present a new normal form, *Conjunctive Query Normal Form* (CQNF) for a conjunctive query in $\mathcal{EL}^{++}$.

Definition 3.4. *A conjunctive query q is in CQNF if*

1. *All nominals are eliminated.*
2. *$\forall x \in Var(q)$, there exists one and only one concept term.*
3. *All concepts are normalized.*
4. *All the conjuncts of a concept term are concept names*
5. *Each role term consists of role names only.*

A conjunctive query $q(\vec{x}) \leftarrow p_1 \wedge p_2 \wedge \cdots \wedge p_i \wedge \cdots \wedge p_n$ can be converted into an equivalent conjunctive query q' in CQNF by recursively applying the following rules:

1. If $p_i = (C_1 \sqcap \cdots \sqcap C_i \sqcap \{a\} \sqcap \cdots \sqcap C_n)(x)$, then replace all the occurrence of x in q with a and discard conjunct $\{a\}$ in p_i.
2. $\forall x \in Var(q)$, replace $\bigwedge_{i=1}^{n} p_i$, where $p_i = C_i(x)$ with $(\prod_{i-1}^{n} C_i)(x)$ i.e. $C(x) \wedge D(x) \wedge E(x) \rightsquigarrow C \sqcap D \sqcap E(x)$.
3. If $\exists$ no $p_i = C_i(x)$, where $1 \leq i \leq n$ and $x \in Var(q)$, then $q'(\vec{x}) \leftarrow p_1 \wedge p_2 \wedge \cdots \wedge p_i \wedge \cdots \wedge p_n \wedge p_{n+1}$ and $p_{n+1} = \top(x)$ i.e. add $\top(x)$ as a new term to q if x doesn't occur in any concept term of q.
4. If $C \notin NORM_{\mathcal{K}}(C)$, where C is a concept in a term, then replace C with $C' \in NORM_{\mathcal{K}}(C)$.
5. If $p_i = (C_1 \sqcap \cdots \sqcap C_{i-1} \sqcap \exists r.D \sqcap C_{i+1} \sqcap \cdots \sqcap C_n)(x)$, then $q'(\vec{x}) \leftarrow p_1 \wedge p_2 \wedge \cdots \wedge p'_i \wedge p''_i \wedge p'''_i \wedge \cdots \wedge p_n$, where $p'_i = (C_1 \sqcap \cdots \sqcap C_{i-1} \sqcap C_{i+1} \sqcap \ldots C_n)(x)$, $p''_i = D(y)$, $p'''_i = r(x, y)$, y is a new variable not occurring in q.

6. If $p_i = r_1 \circ r_2(x, y)$, then $q'(\vec{x}) \leftarrow p_1 \wedge p_2 \wedge \cdots \wedge p_i' \wedge p_i'' \wedge \cdots \wedge p_n$, where $p_i' = r_1(x, z), p_i'' = r_2(z, y)$, z is a new variable not occurring in q.

Removing the complex role chains from the role terms results in role terms consisting of simple role names only. The concepts in a term consists of (conjunction of) concept names only and in none of the terms, both concept and role names occur together. Two queries q, q' are *equivalent* w.r.t. $\mathcal{K}$ if the result sets of the two queries are the same i.e. $q(\mathcal{K}) = q'(\mathcal{K})$. It can also be shown that every CQ has an equivalent CQ in CQNF, because applying the above rules always generate an equivalent query. Example 8 contains a set of queries and their equivalent queries in CQNF.

Example 8. *Consider CQs* $q(x) \leftarrow \exists y.C(x) \wedge D(x) \wedge r(x,y)$,
$q_1(x) \leftarrow E \sqcap \exists r.F(x)$.
$q_2(x) \leftarrow E(x) \wedge F(y) \wedge r_1 \circ r_2(x, y)$.
Then, $q_3(x) \leftarrow \exists y.C \sqcap D(x) \wedge r(x,y) \wedge \top(y)$ *is the CQNF of* $q(x)$.
$q_4(x) \leftarrow \exists y.E(x) \sqcap F(y) \wedge r(x,y)$ *is the CQNF of* $q_1(x)$.
$q_5(x) \leftarrow \exists z.E(x) \sqcap F(y) \wedge r_1(x, z) \wedge r_2(z, y)$ *is the CQNF of* $q_2(x)$.

A set of queries and their equivalent queries in CQNF are provided in Example 8. Since every $\mathcal{EL}^{++}$-CQ has an equivalent CQ in CQNF from now, unless otherwise mentioned we assume that all our CQs are in CQNF. If a query q is in CQNF, any term tm of q will contain a concept name, a conjunction of concept names, or a role name. The set of variables introduced by the rewriting rules is denoted by $NormVar(q)$. The set of variables in the original query (provided by the user) is denoted by $Var_o(q) = Var(q) / NormVar(q)$. While we generate specializations by replacing a variable, we do not replace the variables in $NormVar(q)$ as we eliminate these variables from the queries generated by our method in the post-processing steps.

Pre-processing steps

Before computing the minimal specialization of a conjunctive query q we can extract concept hierarchy, role hierarchy, etc. from the KB which doesn't depend on the given query. So, it is appropriate to perform pre-processing steps to make our method more efficient. Since we know that CQ answering is decidable only for *regular* $\mathcal{EL}^{++}$, we only consider KBs that are regular and in a normal form. Our method utilizes the hierarchies which are generated during the pre-processing steps. The hierarchies computed in the pre-processing steps are:

- A completion graph $\mathcal{C}$ along with a *concept hierarchy* and a *role hierarchy* $\mathcal{H}$

  We first convert the KB $\mathcal{K}$ into a *normal form* [Baader *et al.*, 2005]. then, we can easily generate the concept hierarchy from the normalized TBox. In case of regular $\mathcal{EL}^{++}$, we can also obtain the *partial ordering (hierarchy)* $\mathcal{H}$ over the role names occurring in the normalized RBox using the role inclusions available.

- A *role successor hierarchy* and *role predecessor hierarchy* collection $\mathcal{F}$ as explained below.

Computing Collection $\mathcal{F}$

One way to specialize a query is by replacing a variable x or an individual a from the CQ with another individual b in the ABox. Since the size of the ABox is usually big, its more useful to compute the ordering of all ABox individuals w.r.t. set inclusion of their role predecessor/successors for each role $r \in \mathrm{N}_R^{\mathcal{K}}$. For this purpose, we populate a *collection* $\mathcal{F}$ consisting of *directed acyclic graphs (DGs).* Each directed acyclic graph (DG) is a hierarchy among the individuals occurring in the ABox and corresponds to a role and either to its *predecessor* or to its *successor*. In $r(a,b)$, a is the predecessor and b is the successor. So, if there are N roles in our KB then we obtain 2*N DGs in the collection $\mathcal{F}$. Let

$\mathbf{Succ}(r,a) = \{b \mid r(a,b) \in \mathcal{A}\}$ and $\mathbf{Pred}(r,a) = \{b \mid r(b,a) \in \mathcal{A}\}$.
$\top_i$ represent the auxiliary topmost level individual that contains all the possible values i.e. $\mathbf{Succ}(r,\top_i) = \mathbf{Pred}(r,\top_i) = \{a \mid a \in Indv(\mathcal{A})\}$, $\forall r \in \mathrm{N}_R^{\mathcal{K}}$.
Let $<_p$, $<_s$ represent the ordering relations over individuals occurring in the ABox such that $a <_p b$ if $\mathbf{Pred}(r,a) \subset \mathbf{Pred}(r,b)$ and $a <_s b$ if $\mathbf{Succ}(r,a) \subset \mathbf{Succ}(r,b)$ for $a,b \in \mathcal{A}$.

Then $\mathcal{G}_r^p = (V_r^p, E_r^p)$, $\mathcal{G}_r^s = (V_r^s, E_r^s)$ represent the DGs for a role r and the nodes (which are ABox individuals) are ordered w.r.t. set inclusion of their set of role successors, predecessors of role r respectively. Here

$$E_r^p = \{(a,b) \mid b <_s a\},\ E_r^s = \{(a,b) \mid b <_p a\} \text{ and}$$
$$V_r^p = V_r^s = \{a \in Indv(\mathcal{A})\}$$

In Example 9, the DGs corresponding to a role and an ABox are computed. As we are considering regular RBoxes, there exists a partial ordering among the roles occurring in a RBox and we calculate the collection $\mathcal{F}$ by taking this role ordering also into consideration.

Example 9. *Consider our ABox has the following axioms.*

$$\mathcal{A} = \{A(a),\quad B(b),\quad r(a,b),\quad r(a,c),\quad r(b,c)\}$$

The ordering for the DG $\mathcal{G}_r^p$ is $\{c <_s b <_s a\}$. Here, a is at the top level, and b is in the second level and c is at the bottom level. Similarly, the ordering for DG $\mathcal{G}_r^s$ is $\{a <_p b <_p c\}$. If the ABox has the following axioms also.

$$r_1(c,e) \quad r_1(c,f) \quad r_1(c,g)$$

and if the RBox contains the following assertion : $r_1 \sqsubseteq r$
Then the ordering for the DG $\mathcal{G}_r^p$ is $\{b <_s a <_s c\}$.

We assume that the above two preprocessing steps are performed and $\mathcal{C}$, $\mathcal{H}$, $\mathcal{F}$ are readily available when the KB $\mathcal{K}$ is accessed. In the following sections, unless mentioned otherwise a query q refers to an $\mathcal{EL}^{++}$-conjunctive query in CQNF and $\mathcal{K}$ refers to a normalized and regular $\mathcal{EL}^{++}$-KB. The original knowledge base over which the user posed the query is denoted by $\mathcal{K}^o$ and the set of concept and role names occurring in $\mathcal{K}^o$ is represented by N_C^o and N_R^o.

Minimal concept specializations, the minimal specializations which are obtained by changing one of the concepts occurring in the query are those with the highest potential of being a minimal specialization of the query and now we present a method to calculate these specializations.

3.2.2 Minimal concept specialization

One way to specialize a query is to specialize the concept occurring in a term of the query.

Definition 3.5. *A specialization q' of a query q w.r.t. $\mathcal{K}$ is called a* concept specialization *of q w.r.t. $\mathcal{K}$ if q' is obtained by replacing a concept C in a (concept) term of q with another concept $C' \in \mathrm{N}_C^{\mathcal{K}}$. A concept specialization q' of a query q w.r.t. $\mathcal{K}$ is called a* minimal concept specialization *of q w.r.t. $\mathcal{K}$ if there exists no other concept specialization q'' of q w.r.t. $\mathcal{K}$ such that $q' \subset_{\mathcal{K}} q'' \subset_{\mathcal{K}} q$.*

Let $CSpec_{\mathcal{K}}(q)$ be the set of all concept specializations of a query q w.r.t. $\mathcal{K}$ and $minCSpec_{\mathcal{K}}(q)$ be the set of all minimal concept specializations of a query q w.r.t. $\mathcal{K}$.

A specialization C' of a concept C w.r.t. $\mathcal{K}$ (defined in Section 3.1.1) is called *minimal specialization* of C w.r.t. $\mathcal{K}$ if $\exists$ no other concept C'' such that $C' \sqsubset_{\mathcal{K}} C'' \sqsubset_{\mathcal{K}} C$ and it is represented by $C' \sqsubset_{\mathcal{K}}^{min} C$. $minCSpec_{\mathcal{K}}(C)$ represents the set of all *minimal specializations* of a concept C w.r.t. $\mathcal{K}$. However, replacing a concept C with its minimal specialization w.r.t. $\mathcal{K}$ may not generate a specialization of the query q in every case. Depending on the query q, the new query obtained by replacing C with C' can still have the same set of results and the condition 1 in Definition 3.1 may or may not be satisfied.

So, we define a minimal specialization of a concept C w.r.t. q and $\mathcal{K}$ and a specialization C' of a concept C w.r.t. $\mathcal{K}$ is called a *minimal specialization* w.r.t. q i.e. $C' \sqsubset_{\mathcal{K},q}^{min} C$, if

(a) $\exists$ no concept $C^{''}$ such that $C' \sqsubset_{\mathcal{K}} C^{''} \sqsubset_{\mathcal{K}} C$,

(b) $q'(\mathcal{K}) \neq q(\mathcal{K})$, where q' is obtained by replacing C with C'.

and $minCSpec_{\mathcal{K}}^{q}(C)$ represents the set of all *minimal specializations* of a concept C w.r.t. $\mathcal{K}$ and q. From now, a minimal specialization of a concept C represents a minimal specialization of C w.r.t. the query q and the KB $\mathcal{K}$.

For a concept C there can be more than one minimal specialization as explained in Example 10. So, there can be more than one minimal concept specialization for any given query. Also, if a query q has more than one variable, then any 2 queries (specializations) obtained by replacing two different concepts (in concept terms corresponding to each variable) with their minimal specializations may not be comparable w.r.t. query containment and both these specializations will be minimal concept specializations of q.

Example 10. *Assume a TBox $\mathcal{T}$ with just the following concept inclusions*
$\mathcal{T} = \{\ Father \sqsubseteq Person, \qquad Mother \sqsubseteq Person\ \}$
Then, $q_1(x) \leftarrow \exists y.Father(x) \wedge hasParent(y, x)$
$q_2(x) \leftarrow \exists y.Mother(x) \wedge hasParent(y, x)$ *are incomparable w.r.t. query containment and both are minimal concept specializations of*
$q(x) \leftarrow \exists y.Person(x) \wedge hasParent(x, y)$

In $\mathcal{EL}^{++}$ a concept (without concrete domains) is built according to the syntax rule $C := \top \mid \bot \mid \{a\} \mid A \mid C \sqcap D \mid \exists r.C$, where A ranges over N_C, r ranges over N_R, a ranges over N_I and C, D range over $\mathcal{EL}^{++}$-concepts. Since we convert queries into CQNF, nominals and concepts (conjuncts) of the form $\exists r.A$ do not occur in the query. As, we are interested only in the queries with non-empty result sets, $\bot$ does not occur in our queries. So, for a query in CQNF only the concept specializations of concept names or conjunction of concept names need to be computed.

The concept specialization of a query can be obtained by either adding a new conjunct or by replacing a conjunct with its specialization. To obtain a minimal concept specializations, it is evident that a complex concept can't be added as a conjunct. We can also make use of $\bot$ to avoid generating empty concept specializations (i.e. specializations with an empty result set). Let C be a concept that we are specializing and assume that the TBox has the following two GCIs $D \sqsubseteq_{\mathcal{K}} C$ and $D \sqsubseteq_{\mathcal{K}} \bot$. Then we don't have to consider D as a specialization of C as, the concept specialization obtained will be empty. However, we can obtain nominals as specializations of a concept in some of the cases and they can even be minimal specializations.

Computing Minimal Concept Specializations

Now, we devise a computation procedure for generating all the minimal concept specializations of a query w.r.t. a KB. To this end we introduce few auxiliary sets. Let, $C = C_1 \sqcap C_2 \sqcap \ldots \sqcap C_n$ be a concept occurring in a term $C(x)$ of a query, where x is a variable and $C_1, C_2, \ldots, C_n$ are concept names. Let $ERC_{\mathcal{K}}$ be a set which contains all the concepts of the form $\exists r.B$ where $r \in \mathrm{N}_R^{\mathcal{K}}$ and $B \in \mathrm{N}_C^{\mathcal{K}}$ i.e. $ERC_{\mathcal{K}} = \{\ \exists r.B \mid r \in \mathrm{N}_R^{\mathcal{K}}, B \in \mathrm{N}_C^{\mathcal{K}}\}$.

Since our KB is in normal form, a minimal specialization of a concept name C can be a concept D of the form

1. $D \in \mathrm{N}_C^{\mathcal{K}} \cup ERC_{\mathcal{K}}$
2. $C \sqcap B$, where $B \in \mathrm{N}_C^{\mathcal{K}} \cup ERC_{\mathcal{K}}$
3. $\{a\}$, where $a \in Indv_{\mathcal{K}}(\mathcal{A})$.

As, $C \sqcap \{a\} \equiv \{a\}$, we don't consider adding a nominal as a conjunct to the concept name C to obtain a minimal specialization of C. Here, $Indv_{\mathcal{K}}(C) = \{a \in \mathcal{A} \mid \mathcal{A} \models C(a)\}$.

In addition, a minimal specialization of a concept C is not subsumed by any other minimal specialization of C, and it wouldn't generate queries having the same result set. So, we first compute these concepts which generate queries with the same result set and eliminate them from the set $\mathrm{N}_C^{\mathcal{K}} \cup ERC_{\mathcal{K}} \cup Indv_{\mathcal{K}}(C)$. For this purpose we define the set

$EQUAL^q_{\mathcal{K}}(C) = \{D \in \mathrm{N}_C^{\mathcal{K}} \cup ERC_{\mathcal{K}} \cup Indv_{\mathcal{K}}(C) \mid q'(\mathcal{K}) = q(\mathcal{K})$, where q' is obtained from q either by replacing C with D or adding D as a conjunct to C $\}$

It is the set of concepts which either on replacing C or on adding as a conjunct to C generate queries (non-specializations) that are equivalent to the original query (see Example 11). Let $IC^q_{\mathcal{K}} = (\mathrm{N}_C^{\mathcal{K}} \cup ERC_{\mathcal{K}}) \setminus EQUAL^q_{\mathcal{K}}(C)$.

Example 11. *Assume a KB $\mathcal{K}$ where*
$\mathcal{T} = \{$ *Father* $\sqsubseteq$ *Person*$\}$
$\mathcal{A} = \{$ *Father(b), Father(c) , Person(a), livesIn(b, e), livesIn(c, b)* $\}$
and a query $q_1(x) \leftarrow \exists y.$*Person(x)* $\wedge$ *livesIn*(x, y)
Then, the query $q_2(x) \leftarrow \exists y.$*Father*$(x) \wedge$ *livesIn*(x, y)
is obtained by replacing Person with Father. However $q_1(\mathcal{K}) = q_2(\mathcal{K})$ *and* q_2 *is not a specialization of* q_1*. Here,* $Father \in EQUAL^q_{\mathcal{K}}(Person)$.

Computing the 3 forms of minimal specializations of a concept name

From the remaining concepts in $\mathrm{N}_C^{\mathcal{K}} \cup ERC_{\mathcal{K}} \cup Indv_{\mathcal{K}}(C)$, we then select the minimal specializations (in the above 3 forms) of the concept C. We define

$DS^q_{\mathcal{K}}(C) = \{D \in IC^q_{\mathcal{K}} \mid D \sqsubset_{\mathcal{K}} C \wedge \exists$ no $E \in IC^q_{\mathcal{K}}$: $D \sqsubset_{\mathcal{K}} E \wedge E \sqsubset_{\mathcal{K}} C\} \setminus$
$\{D \in IC^q_{\mathcal{K}} \mid D \sqsubseteq_{\mathcal{K}} \bot\}$
It is the set of concepts that (a) are directly subsumed by the concept C and, (b) generate a concept specialization when C is replaced by them in the query.

$NC^q_{\mathcal{K}}(C) = \{D \in IC^q_{\mathcal{K}} \mid C \not\sqsubseteq_{\mathcal{K}} D \wedge D \not\sqsubseteq_{\mathcal{K}} C \wedge \exists$ no $E \in IC^q_{\mathcal{K}}$ such that
$C \not\sqsubseteq_{\mathcal{K}} E \wedge E \not\sqsubseteq_{\mathcal{K}} C \wedge D \sqsubset_{\mathcal{K}} E\} \setminus \{D \in IC^q_{\mathcal{K}} \mid D \sqsubseteq_{\mathcal{K}} \bot\}$
It is the set of concepts that neither are subsumed by C nor subsume C w.r.t. $\mathcal{K}$ and when added to C as a conjunct generates a specialization.

$DNS^q_{\mathcal{K}}(C) = \{a \in Indv_{\mathcal{K}}(C) \mid \mathcal{A} \not\models D(a), \forall D \in DS^q_{\mathcal{K}}(C) \cup NC^q_{\mathcal{K}}(C)\}$
It is the set of nominals which are minimal specializations of C.

Minimal specializations of C, a conjunction of concept name(s):

Finally, we denote $minConcSpec^q_{\mathcal{K}}(C)$ as the set

$$minConcSpec^q_{\mathcal{K}}(C) = \begin{cases} \{\ D\ \in DS^q_{\mathcal{K}}(C) \cup DNS^q_{\mathcal{K}}(C)\ \} \cup \\ \{\ C \sqcap D \mid\ D \in NC^q_{\mathcal{K}}(C)\ \},\ C \text{ is atomic ;} \\ \{C_1 \sqcap \cdots \sqcap minConcSpec^q_{\mathcal{K}}(C_i) \sqcap \cdots \sqcap C_n\}, \text{ where} \\ \ C_1 \sqcap \cdots \sqcap C_{i-1} \sqcap C_{i+1} \cdots \sqcap C_n \not\equiv_{\mathcal{K}} minConcSpec^q_{\mathcal{K}}(C_i). \end{cases}$$

We considered $C = C_1 \sqcap \ldots \sqcap C_n$ if C is not atomic. When we compute the minimal specializations of a concept C irrespective of the query, the set $EQUAL^q_{\mathcal{K}}(C) = \emptyset$.

We now define the set
$\mathbf{minConcSpec}_{\mathcal{K}}(q) = \{q' |\ q'(\mathcal{K}) \neq \emptyset$, where q' is obtained by replacing a concept C in q with $C' \in minConcSpec^q_{\mathcal{K}}(C)\ \}$
and Lemma 3.1 shows that the set $\mathbf{minConcSpec}_{\mathcal{K}}(q)$ contains all the minimal concept specializations of a query q w.r.t. $\mathcal{K}$.

Lemma 3.1. *Given a query q and a KB $\mathcal{K}$,*
(1) $q' \in \boldsymbol{minConcSpec}_{\mathcal{K}}(q) \supseteq minCSpec_{\mathcal{K}}(q)$ and
(2) $q' \in \boldsymbol{minConcSpec}_{\mathcal{K}}(q) \subseteq Spec_{\mathcal{K}}(q)$.

Proof. Claim (1)
Since our query q is in CQNF, there exists only one concept term for each variable $x \in Var(q)$. To obtain a minimal concept specialization it is sufficient to replace a concept C of a concept term in q with its minimal specialization.
i.e. it is sufficient to prove that $minConcSpec^q_{\mathcal{K}}(C)$ contains all the minimal specializations of C. (As every other concept can be treated similarly, we can generate all the minimal concept specializations of q from $\mathbf{minConcSpec}_{\mathcal{K}}(q)$.)
This implies, if concept $C_a \notin minConcSpec^q_{\mathcal{K}}(C)$ then C_a is not a minimal specialization of C. We show it with proof by contradiction and induction.

Assume, there exists an arbitrary $\mathcal{EL}^{++}$-concept $C_a \in minCSpec^q_{\mathcal{K}}(C)$ and $C_a \notin minConcSpec^q_{\mathcal{K}}$(C). (1)
case (i) $C_a = \top$, $\top$ can't be a minimal specialization and we reach a contradiction.
case (ii) $C_a = \bot$, $\bot$ too can't be a minimal specialization of C and we reach a contradiction.
case (iii) $C_a = A$, concept name
(a) If C is a concept name
then $C_a \in DS^q_{\mathcal{K}}(C)$ as both C, C_a are concept names and $C_a \in minConcSpec^q_{\mathcal{K}}(C)$ which is a contradiction.
(b)If C is of the form $C_1 \sqcap \ldots \sqcap C_i \sqcap \ldots \sqcap C_n$
then $C_a \sqsubset_{\mathcal{K}} C_i$ for some i and $C_1 \sqcap \ldots \sqcap C_a \sqcap \ldots \sqcap C_n \sqsubset_{\mathcal{K}} C$.
Since $C_a \sqsubset_{\mathcal{K}} C \wedge C_a \sqsubset_{\mathcal{K}} C_i$, $C_a \equiv_{\mathcal{K}} C_1 \sqcap \ldots \sqcap C_a \sqcap \ldots \sqcap C_n \sqsubset_{\mathcal{K}} C$
Then, either $C_a \in minConcSpec_{\mathcal{K}}(C)$ or C_a is not a minimal concept specialization of C and we obtain a contradiction.
case (iv) $C_a = \{a\}$, a nominal
(a) C is a concept name
Since $C_a \notin minConcSpec^q_{\mathcal{K}}(C)$, we have $\{a\} \notin DNS^q_{\mathcal{K}}(C)$.
So, there exists a concept $D \in DS^q_{\mathcal{K}}(C) \cup NC^q_{\mathcal{K}}(C)$ such that
$\mathcal{A} \models D(a) \wedge \{a\} \sqsubset_{\mathcal{K}} D \sqsubset_{\mathcal{K}} C \wedge D \in DS^q_{\mathcal{K}}(C) \cup NC^q_{\mathcal{K}}(C)$ (2)
From (1) and (2), we obtain a contradiction.
(b) C is non-atomic, as in case (iii)
case (v) $C_a = \exists r.A$, where $r \in \mathrm{N}^{\mathcal{K}}_R$ and $A \in \mathrm{N}^{\mathcal{K}}_C$.

Similar to case (iii)
case (vi) $C_a = D \sqcap E$ (D, E are $\mathcal{EL}^{++}$-concepts)
We assumed that $D \sqcap E \sqsubset^{min}_{\mathcal{K},q} C$ and $D \sqcap E \notin minConcSpec^q_{\mathcal{K}}(C)$.
Then, $D \sqcap E \equiv_{\mathcal{K}} D \sqcap E \sqcap C \sqsubset_{\mathcal{K}} C \sqcap C \equiv_{\mathcal{K}} C$.
Since we are interested only in minimal specializations of C, only the following are possible: $D \sqcap C \sqsubset^{min}_{\mathcal{K},q}$ C, $E \sqcap C \sqsubset^{min}_{\mathcal{K},q} C$.
(as $D \sqcap E \sqcap C \sqsubset_{\mathcal{K}} D \sqcap C \sqsubset_{\mathcal{K}} C$ and $D \sqcap E \sqcap C \sqsubset_{\mathcal{K}} E \sqcap C \sqsubset_{\mathcal{K}} C$)
W.l.o.g. let us assume that $C_a \equiv D \sqcap C \sqsubset^{min}_{\mathcal{K},q} C$.
We can further divide D into $D_1 \sqcap \mathrm{E}_1$ and continue this for k steps until the num. of conjuncts in D_k is 1 and $D_k \sqcap C \sqsubset^{min}_{\mathcal{K},q}$C.
Here $D_k \not\sqsubseteq_{\mathcal{K}}$ C, else if $D_k \sqsubset_{\mathcal{K}}$ C, then $D_k \in DS^q_{\mathcal{K}}(C) \in minConcSpec^q_{\mathcal{K}}(C)$
As D_k doesn't subsume C we can infer that $C \not\sqsubseteq_{\mathcal{K}} D_k$ and also $D_k \not\sqsubseteq_{\mathcal{K}}$C
Then, $D_k \sqcap C \in NC^q_{\mathcal{K}}(C) \in minConcSpec^q_{\mathcal{K}}(C)$ which is a contradiction.

So, $minConcSpec^q_{\mathcal{K}}(C)$ contains all the minimal specializations of a concept C.
Hence, $\mathbf{minConcSpec}_{\mathcal{K}}(q)$ contains all minimal concept specializations of q w.r.t. $\mathcal{K}$.

Claim (2): Proof by contradiction.
Assume, $\exists$ a query $q_1 \in \mathbf{minConcSpec}_{\mathcal{K}}(q)$ and $q_1 \notin Spec_{\mathcal{K}}(q)$.
Then q_1 will not satisfy atleast one of the conditions in Definition 3.1.
Condition 1: All the minimal specializations $C' \in minConcSpec^q_{\mathcal{K}}(C)$ of a concept C only generate queries which are strictly contained in q. So, condition 1 is always satisfied.
Condition 2: If $q' \in \mathbf{minConcSpec}_{\mathcal{K}}(q)$ then $q'(\mathcal{K}) \neq \emptyset$ and the condition is directly satisfied.
Condition 3: Since we replace only one concept C with its minimal specialization, we always satisfy condition 3a which is sufficient to satisfy condition 3.
Condition 4: Once again, since we replace only a concept C the number of variables remain unchanged. So, condition 4 is also satisfied.

Hence, every query $q_1 \in \mathbf{minConcSpec}_{\mathcal{K}}(q)$ is a specialization of q w.r.t. $\mathcal{K}$. □

3.2.3 Minimal role specialization

A role specialization is obtained by replacing a role r with its specialization r' w.r.t. $\mathcal{K}$. Role specializations of a query can be obtained in a similar way to concept specializations.

Definition 3.6. *A specialization q' of q w.r.t. $\mathcal{K}$ is called a* role specialization *of q w.r.t. $\mathcal{K}$ if q' is obtained by replacing a role r of the role term $r(v_1, v_2)$ in q with a role $r' \in \mathrm{N}^{\mathcal{K}}_R$. A role specialization q_1 of q w.r.t. $\mathcal{K}$ is* minimal *if there exists no other role specialization q_2 of q w.r.t. $\mathcal{K}$ such that $q_1 \subset_{\mathcal{K}} q_2$ and $q_2 \subset_{\mathcal{K}} q$.*

We define $RSpec_{\mathcal{K}}(q)$ as the set of all role specializations of a query q w.r.t.

$\mathcal{K}$ and $minRSpec_{\mathcal{K}}(q)$ as the set of all minimal role specializations of query q w.r.t. $\mathcal{K}$. Example 12 shows how role specializations w.r.t. some KB can be obtained.

Example 12. *Let the TBox of KB $\mathcal{K}$ contain the following SRI*
$\mathcal{R} = \{hasFather \sqsubseteq hasParent\ \}$
Among the queries
$q(x) := \exists y.Man(x) \wedge hasParent(x, y)$
$q_2(x) := \exists y.Man(x) \wedge hasFather(x, y)$
we can see that $q_2(x)$ is a role specialization of $q_1(x)$ w.r.t. $\mathcal{K}$.

A specialization r' of a role r w.r.t. $\mathcal{K}$ (defined in Section 3.1.1) is called a *minimal specialization* of r w.r.t. $\mathcal{K}$ i.e. $r' \sqsubset^{min}_{\mathcal{K}} r$ if there exists no other specialization r'' of r such that $r' \sqsubset_{\mathcal{K}} r'' \sqsubset_{\mathcal{K}} r$. $minRSpec_{\mathcal{K}}(r)$ represents the set of *minimal specializations* of a role r w.r.t. $\mathcal{K}$. As explain in Section 3.2.2 we also need define minimal specializations of a role w.r.t. $\mathcal{K}$ and q.

A specialization r_1 of a role r w.r.t. $\mathcal{K}$ is a *minimal specialization* of r w.r.t. q and $\mathcal{K}$ i.e. $r_1 \sqsubset^{min}_{\mathcal{K},q} r$, if
(a) $\exists$ no other specialization r'' of r w.r.t. $\mathcal{K}$ such that $r' \sqsubset_{\mathcal{K}} r'' \sqsubset_{\mathcal{K}} r$,
(b) q' obtained by replacing r with r' and $q'(\mathcal{K}) \neq q(\mathcal{K})$.
and $minRSpec^{q}_{\mathcal{K}}(r)$ represents the set of *minimal specializations* of a role r w.r.t. $\mathcal{K}$ and q.

Since a query (in CQNF) doesn't have conjuncts of the form $\exists r.D$ in a concept term, we do not have to consider specializing a role in a concept term. Moreover, since our queries are in CQNF, only simple role names occur in our queries.

Computing Minimal Role Specializations

Now, we devise a computation procedure for generating all the minimal role specializations of a query w.r.t. $\mathcal{K}$. To this end we introduce few auxiliary sets. Since, we normalize the RBox $\mathcal{R}$ in the pre-processing steps, all the role inclusions in $\mathcal{R}$ are of the from $r \sqsubseteq s$ or $r_1 \circ r_2 \sqsubseteq r$. The minimal specializations of a role name r are of the form (a) r_1, or (b) $r_1 \circ r_2$, where $r_1, r_2 \in \mathrm{N}^{\mathcal{K}}_R\}$. $RC_{\mathcal{K}}$ is the set of all possible binary role compositions in $\mathcal{K}$ i.e. $RC_{\mathcal{K}} = \{r_1 \circ r_2 \mid r_1, r_2 \in \mathrm{N}^{\mathcal{K}}_R\}$.

In addition, a minimal specialization of a role r is not subsumed by any other specialization of r, and it wouldn't generate queries having the same result set. So, we first compute these roles which generate queries with the same result set and eliminate them from the set $\mathrm{N}^{\mathcal{K}}_R \cup RC_{\mathcal{K}}$. For this purpose we define the set

$EQUAL^{q}_{\mathcal{K}}(r) = \{r' \in N^{\mathcal{K}}_R \cup RC_{\mathcal{K}} \mid q'(\mathcal{K}) = q(\mathcal{K})$, where q' is obtained from q by replacing r with r' $\}$.
This set of roles generates queries (non-specializations) that are equivalent to the original query if r is replaced in q by an element.

Let $ICR^q_{\mathcal{K}} = (\mathrm{N}^{\mathcal{K}}_R \cup RC_{\mathcal{K}}) \setminus EQUAL^q_{\mathcal{K}}(r)$.

From the remaining roles in $\mathrm{N}^{\mathcal{K}}_R \cup RC_{\mathcal{K}}$ we then select the minimal specializations of the role r and we define the set $minRoleSpec^q_{\mathcal{K}}(r)$ as

$$minRoleSpec^q_{\mathcal{K}}(r) = \{r_1 \in ICR^q_{\mathcal{K}} \mid r_1 \sqsubset_{\mathcal{K}} r \wedge \nexists\ r_2 \in ICR^q_{\mathcal{K}} : r_1 \sqsubset_{\mathcal{K}} r_2 \sqsubset_{\mathcal{K}} r\}.$$

This set consists of all the minimal specializations of a role name r. When we compute the minimal specializations of a role r irrespective of the query, the set $EQUAL^q_{\mathcal{K}}(r) = \emptyset$.

The set $\mathbf{minRoleSpec}_{\mathcal{K}}(q)$ is obtained as
$\mathbf{minRoleSpec}_{\mathcal{K}}(q) = \{q' \mid q'(\mathcal{K}) \neq \emptyset$, where q' is obtained by replacing a role r in q by a role $r_1 \in minRoleSpec_{\mathcal{K}}(r)\ \}$
and in Lemma 3.2 we show that $\mathbf{minRoleSpec}_{\mathcal{K}}(q)$ contains all the minimal role specializations of q w.r.t. $\mathcal{K}$.

Lemma 3.2. *Given a query q and a KB $\mathcal{K}$,*
(1) $\boldsymbol{minRoleSpec}_{\mathcal{K}}(q) \supseteq minRSpec_{\mathcal{K}}(q)$ and,
(2) $\boldsymbol{minRoleSpec}_{\mathcal{K}}(q) \subseteq Spec_{\mathcal{K}}(q)$.

Proof. Claim (1)
Similar to the argument in Lemma 3.1, we can conclude that, all the minimal role specializations of a query q w.r.t. $\mathcal{K}$ are in $\mathbf{minRoleSpec}_{\mathcal{K}}(q)$ when $minRoleSpec^q_{\mathcal{K}}(r)$ contains all the minimal specializations of role r w.r.t. q.

Using proof by contradiction and proof by induction.
Assume, $\exists$ a minimal specialization r' of r and $r_1 \notin minRoleSpec^q_{\mathcal{K}}(r)$ (1)
case(i) r' be a role name i.e. $r' \in \mathrm{N}^{\mathcal{K}}_R$.
As $r' \notin minRoleSpec^q_{\mathcal{K}}(r)$, $\exists$ a role $r'' \in ICR^q_{\mathcal{K}} : r' \sqsubset_{\mathcal{K}} r'' \sqsubset_{\mathcal{K}} r$. Then, r' is not a minimal specialization of role r, and we obtain a contradiction.
case (ii) $r' \equiv_{\mathcal{K}} r_1 \circ r_2$ and $r_1, r_2 \in \mathrm{N}^{\mathcal{K}}_R$ i.e. $r' \in ICR^q_{\mathcal{K}}$
As $r' \notin minRoleSpec^q_{\mathcal{K}}(r)$, $\exists$ a role $r'' \in ICR^q_{\mathcal{K}} : r' \sqsubset_{\mathcal{K}} r'' \sqsubset_{\mathcal{K}} r$ and r' is not a minimal specialization of role r which is a contradiction.
case (iii) $r' \equiv_{\mathcal{K}} r_1 \circ \cdots \circ r_n$ and $r_1, \ldots, r_n \in \mathrm{N}^{\mathcal{K}}_R, n \geq 3$.
We prove that for any random value of $n \geq 3$, r' is not a minimal specialization of r.
Let $n = k + 1$, for some $k \geq 2$ i.e. $r_1 \circ r_2 \circ \cdots \circ r_k \circ r_{k+1} \sqsubset_{\mathcal{K}} r$.
Since, $\mathcal{K}$ is normalized, (w.l.o.g. assume that) $\exists$ a role $r'_k \in \mathrm{N}^{\mathcal{K}}_R$ such that $r_k \circ r_{k+1} \sqsubset_{\mathcal{K}} r'_k$.
This implies $r_1 \circ r_2 \circ \cdots \circ r_k \circ r_{k+1} \sqsubset_{\mathcal{K}} r_1 \circ r_2 \circ \cdots \circ r'_k \sqsubset_{\mathcal{K}} r$ and $r_1 \circ r_2 \circ \cdots \circ r_k \circ r_{k+1} \not\sqsubset^{min}_{\mathcal{K},q} r$ which is a contradiction.
Therefore, when $n \geq 3$, r' cannot be a minimal specialization of r.
i.e. $minRoleSpec^q_{\mathcal{K}}(r)$ contains all the minimal specializations of role r.
So, $\mathbf{minRoleSpec}(q,\mathcal{K})$ contains all the minimal role specializations of a query q w.r.t. $\mathcal{K}$.

Claim (2): Proof by contradiction.
Assume, $\exists$ a query $q_1 \in \mathbf{minRoleSpec}_{\mathcal{K}}(q)$ and $q_1 \notin Spec_{\mathcal{K}}(q)$.
Then q_1 will not satisfy atleast one of the conditions in Definition 3.1.
Condition 1: All the minimal specializations $r' \in minRoleSpec^{q}_{\mathcal{K}}(r)$ of a role r only generate queries which are strictly contained in q. So, condition 1 is always satisfied.
Condition 2: If $q' \in \mathbf{minRoleSpec}_{\mathcal{K}}(q)$ then $q'(\mathcal{K}) \neq \emptyset$ and the condition is directly satisfied.
Condition 3: Since we replace only one role r with its minimal specialization, we satisfy condition 3a which is sufficient to satisfy condition 3.
Condition 4: Once again, since we replace only a role r the number of variables remain unchanged. So, condition 4 is also satisfied.

Hence, each query in $\mathbf{minRoleSpec}_{\mathcal{K}}(q)$ is a specialization of q w.r.t. $\mathcal{K}$. □

3.2.4 Minimal conjunct specialization

As described in Section 3.1, we can specialize a query by adding a new role term to the query. We use the role orderings (hierarchies) in $\mathcal{H}$ to add a new role term to the query. These kinds of specializations obtained by adding a new role term are called conjunct specializations.

Definition 3.7. *A specialization q' of a query q w.r.t. $\mathcal{K}$ is called a* conjunct specialization *of q w.r.t. $\mathcal{K}$ if q' is obtained by adding a role term $r_1(u, v)$ as a conjunct to q, where $r_1 \in \mathrm{N}_R$. A conjunct specialization q_1 of q w.r.t. $\mathcal{K}$ is* minimal *if $\exists$ no other conjunct specialization q_2 of q w.r.t. $\mathcal{K}$ such that $q_1 \subset_{\mathcal{K}} q_2 \subset_{\mathcal{K}} q$.*

The set of all minimal conjunct specializations which are equivalent to none of the role specializations are defined by set $minRConjSpec_{\mathcal{K}}(q)$. The set of all role terms which on adding to q generate a specialization in $minRConjSpec_{\mathcal{K}}(q)$ are defined by the set $minRConj_{\mathcal{K}}(q)$.

Example 13. *Let the ABox of KB $\mathcal{K}$ contain the following axioms*
$\mathcal{A} = \{r(a, b), r_1(a, b), r(b, c), r_1(b, c), r(d, e), r_1(g, e), C(a), C(b)\}$
and a query $q(x) := C(x) \wedge r(x, y)$
Then, the query $q_1(x) := C(x) \wedge r(x, y) \wedge r_1(x, y)$
is not a specialization of q as $q(\mathcal{K}) = q_1(\mathcal{K}) = \{a, b\}$.

Computing Minimal Conjunct Specializations

Now, we devise a computation procedure for generating all the minimal conjunct specializations of a query w.r.t. $\mathcal{K}$. To this end we introduce few auxiliary sets $InCompRole_{\mathcal{K}}(r)$ and $minConj_{\mathcal{K}}(q)$. $RC_{\mathcal{K}}$ introduced in the previous section is also used here.

As shown in Example 13, we should look out for the terms which do not reduce

the result set of the new query and omit them in our computation method. For this purpose, we define
$EQUAL\_CONJ^q_{\mathcal{K}}(r) = \{r_1 \in \mathrm{N}^{\mathcal{K}}_R \cup RC_{\mathcal{K}} \mid q'(\mathcal{K}) = q(\mathcal{K})$, where q' is
obtained by adding a role term $r_1(u, v)$ and $\exists$ a role term $r(u, v)$ in $q\}$
It is the set of roles which produce non-specializations w.r.t. a role r.
Let $INCR^q_{\mathcal{K}} = (\mathrm{N}^{\mathcal{K}}_R \cup RC_{\mathcal{K}}) \setminus EQUAL\_CONJ^q_{\mathcal{K}}(r)$

Then we define the set
$InCompRole^q_{\mathcal{K}}(r) = \{r_1 \in INCR^q_{\mathcal{K}} \mid r_1 \not\sqsubseteq_{\mathcal{K}} r$ and $r \not\sqsubseteq_{\mathcal{K}} r_1\}$
This is the set of roles that neither are subsumed by nor subsume r w.r.t. $\mathcal{K}$

The set $minConj_{\mathcal{K}}(q)$ is denoted as
$minConj_{\mathcal{K}}(q) = \{r_1(u, v) \mid r_1 \in InCompRole^q_{\mathcal{K}}(r)$, where $r(u, v) \in q$
$\wedge\, r_1(u, v) \notin q \wedge \exists$ *no* $r_2 \in InCompRole^q_{\mathcal{K}}(r) : r_1 \sqsubset_{\mathcal{K}} r_2\}$
and it contains all the role conjuncts which can be added to the query to obtain a specialization.

The set $\mathbf{minConjSpec}_{\mathcal{K}}(q)$ is obtained as
$\mathbf{minConjSpec}_{\mathcal{K}}(q) = \{q_1 |\ q_1(\mathcal{K}) \neq \emptyset$, where q_1 is obtained by adding to q a
role conjunct $r(u, v) \in minConj_{\mathcal{K}}(q)\}$.
In Lemma 3.3 we show that $\mathbf{minConjSpec}_{\mathcal{K}}(q)$ contains all the minimal conjunct specializations of a query q w.r.t. $\mathcal{K}$ which are not equivalent to any role specialization of q w.r.t. $\mathcal{K}$.

Lemma 3.3. *Given a query q and a KB $\mathcal{K}$,*
(1) $\mathbf{minConjSpec}_{\mathcal{K}}(q) \supseteq minRConjSpec_{\mathcal{K}}(q)$,
(2) $\mathbf{minConjSpec}_{\mathcal{K}}(q) \subseteq Spec_{\mathcal{K}}(q)$.

Proof. Claim (1)
(As in Lemma 3.1 and 3.2) It is sufficient to prove that $minConj_{\mathcal{K}}(q)$ contains all conjuncts required to generate every minimal conjunct specializations of q w.r.t. $\mathcal{K}$ is not equivalent to any role specialization of q w.r.t. $\mathcal{K}$.
i.e. $r(u, v) \in minRConj_{\mathcal{K}}(q) \rightarrow r(u, v) \in minConj_{\mathcal{K}}(q)$.

Proof by contradiction. Assume, $\exists$ a term $r(u, v) \in minRConj_{\mathcal{K}}(q)$ and $r(u, v) \notin minConj_{\mathcal{K}}(q)$
case (i) $\exists$ a role term $r_1(u, v)$ in q.
Then adding $r(u, v)$ to q where $r_1 \sqsubseteq_{\mathcal{K}} r$ or $r \sqsubseteq_{\mathcal{K}} r_1$ would not generate any new minimal specializations.
(a) $r_1 \sqsubseteq_{\mathcal{K}} r$. The generated query is not a specialization.
(b) $r \sqsubseteq_{\mathcal{K}} r_1$, similar to role specialization where r_1 is replaced by r.
So, $r(x, y) \in minRConj_{\mathcal{K}}(q)$ only if $r_1 \not\sqsubseteq_{\mathcal{K}} r \wedge r \not\sqsubseteq_{\mathcal{K}} r_1$ which implies that $r(u, v) \in minRoleConj_{\mathcal{K}}(q)$ and we obtain a contradiction.
case (ii) $\exists$ a no role term $r_1(u, v)$ in q.
Even though $r(u, v)$ is a new restriction on the variable(s) u and/or v, the query obtained by adding $r(u, v)$ as a conjunct isn't a specialization.
So, $r(x, y) \notin minRConj_{\mathcal{K}}(q)$ and we obtain a contradiction

Hence, $\mathbf{minConjSpec}_{\mathcal{K}}(q)$ contains all the minimal role conjunct specializations of a query q w.r.t. $\mathcal{K}$ which are not equivalent to any role specialization of q w.r.t. $\mathcal{K}$.

Claim (2): Proof by contradiction.
Assume, $\exists$ a query $q_1 \in \mathbf{minConjSpec}_{\mathcal{K}}(q)$ and $q_1 \notin Spec_{\mathcal{K}}(q)$.
Then q_1 will not satisfy atleast one of the conditions in Definition 3.1.
Condition 1: All the terms $r'(x, y) \in minConj_{\mathcal{K}}(q)$ only generate queries which are strictly contained in q. So, condition 1 is always satisfied.
Condition 2: If $q' \in \mathbf{minConjSpec}_{\mathcal{K}}(q)$ then $q'(\mathcal{K}) \neq \emptyset$ and the condition is directly satisfied.
Condition 3: Since we add a role term $r'(u, v)$ only when a role term $r(u, v)$ exists in q, we satisfy condition 3b which is sufficient to satisfy condition 3.
Condition 4: Once again, since we add a role term $r'(u, v)$ and both u, v already occur in q the number of variables remain unchanged. So, condition 4 is also satisfied.

Hence, each query in $\mathbf{minConjSpec}_{\mathcal{K}}(q)$ is a specialization of q w.r.t. $\mathcal{K}$. □

3.2.5 Minimal variable specialization

A query q contains a set of variables represented by $Var(q)$ and the set of variables in the query before normalization is represented by the set $Var_o(q)$. $NonDistVar(q)$, $DistVar(q)$ represent the set of non-distinguished and distinguished variables respectively in the query before normalization i.e. $DistVar(q) \cup NonDistVar(q) = Var_o(q)$. By replacing a variable $x \in Var_o(q)$ in query q with another variable $y \in Var_o(q)$ we can obtain a specialization.

Definition 3.8. *A specialization q' of a query q w.r.t. $\mathcal{K}$ is called a* variable specialization *of q w.r.t. $\mathcal{K}$ if q' is obtained from q by replacing a variable x with another variable y, where $x, y \in Var_o(q)$. A variable specialization q' of a query q is called* minimal variable specialization *if $\exists$ no other variable specialization q'' such that $q' \subset_{\mathcal{K}} q'' \subset_{\mathcal{K}} q$.*

The set of all variable specializations is defined as $VSpec_{\mathcal{K}}(q)$. The set of all minimal variable specializations is defined as $minVSpec_{\mathcal{K}}(q)$. In addition, replacing x with an individual a from the ABox can also produce a specialization which is covered in the next section. Specializations obtained by replacing more than one occurrences of the variable x are not specializations (since in a specialization only one syntactic element is changed). But, when only one of the occurrences of the variable x is replaced with another variable y we can no longer guarantee that a specialization will be obtained as shown in Example 14.

Example 14. *Let ABox contain*

$$\mathcal{A} = \{\ isMother(a, b) \quad isFather(b, c) \quad isFather(b, e),$$
$$isMother(f, g) \quad isFather(g, h) \quad C(a) \quad C(f) \quad C(c)\ \}$$

From $q(x) \leftarrow \exists y \exists z.isMother(x,y) \wedge isFather(y,z) \wedge C(z) \wedge \top(x)$
$q_1(x) \leftarrow \exists y.isMother(x,y) \wedge isFather(y,x) \wedge C(x)$
$q_2(x) \leftarrow \exists y.\exists z.isMother(x,y) \wedge isFather(y,z) \wedge C(x).$
we can see that query q has 1 result, query q_1 *has 0 results, but query* q_2 *has 2 results.*

Also, we should treat the distinguished and non-distinguished variables differently as distinguished variables are of higher importance. Depending on the KB (especially ABox) considered, a replacement can generate a specialization or a non-specialization.

Computing Minimal Variable Specialization

We first compute all variable specializations of a query and then eliminate the ones which are contained in other variable specializations. So, the main step is the computation of all the variable specializations of a query.

Example 15. *Consider a query*
$q(x,y) \leftarrow \exists v.C(x) \wedge D(y) \wedge E(u) \wedge r(x,y) \wedge r_1(y,z) \wedge r_2(u,v)$
Then, $connected(x) = \{y\}, connected(y) = \{x,z\}, connected(z) = \{y\}$, $connected(u) = \{v\}, connected(v) = \{u\}$

$depends(x,z) = connected(connected(x)) = connected(\{y\}) = 1$,
$depends(x,y) = 1, depends(x,u) = 0, depends(u,v) = 1.$

$Type(x) = C \sqcap domain(r), Type(y) = D \sqcap range(r) \sqcap domain(r_2)$,
$Type(z) = range(r_1), Type(u) = E \sqcap domain(r_2), Type(v) = range(r_2).$

If a query q has a role term $r(u,v)$, then we say that u is *connected* to v and vice-versa. For a variable $x \in Var(q)$, $connected(x)$ represents the set of variables in $Var(q)$ that are *connected* to x. x is *distantly connected* to y or vice-versa if $x \in connected(connected(\cdots_{(n-1)times}\ connected(y)\cdots))$ (i.e. $x \in connected^n(y)$), where $n \geq 1$ as shown in Example 15. Intuitively, x is *distantly connected* to y if restricting the values of x can also restrict the values of y. $depends(x,y)$ specifies that x is *distantly connected* to y i.e. $x \in connected^n(y)$, $n \geq 1$ and vice versa. If x is distantly connected to y, then $depends(x,y) = 1$ else $depends(x,y) = 0$.

As shown in Example 15, $Type(x)$ is obtained by the conjunction of the concepts C, $domain(r)$ and $range(r)$, where $C(x)$, $r(x,v)$ and $r(u,x)$ are terms in q.

$P(x)$ is the set of possible values of $x \in Var(q)$ in q i.e. $P(x) = \{a \mid a \in q(x)\}$.
$P(x)'$ is the set of possible values of $x \in Var(q')$ in q' i.e. $P(x)' = \{a \mid a \in q'(x)\}$.

Let $varRepl_{\mathcal{K}}(q,x,y)$ be the set that contains only those replacements of x which satisfy the below conditions.

- if $x \in DistVar(q)$

1. x has atleast one other occurrence.
2. $\exists u_i \in connected(x) : depends(u_i, y) \wedge P(x) \supset P(x)'$. $P(x)'$ is the possible values of x after replacement.

- if $x \in NonDist(q)$

1. x also occurs in another term tm.
2. $\neg depends(u_i, z) \vee P(z)' \subset P(z), \forall z \in DistVar(q), \forall u_i \in connected(x)$.
3. q', the query obtained from the replacement should have a reduced result set i.e. $q'(\mathcal{K}) \neq q(\mathcal{K})$.

When the values of $x \in DistVar(q)$ reduces after the replacement, the obtained query will always be strictly contained in the original query. So, we don't check for query containment if $x \in DistVar(q)$.

We generate every variable specialization by performing only the replacements in $varRepl_{\mathcal{K}}(q,x,y)$ for each $x, y \in Var_o(q)$.
$\mathbf{VarSpec}_{\mathcal{K}}(q) = \{q_1 \mid q_1(\mathcal{K}) \neq \emptyset$, where q_1 is obtained by the replacement $r \in varRepl_{\mathcal{K}}(q,x,y)\ \forall x, y \in Var_o(q)\}$.

Lemma 3.4. *Given a query q and a KB $\mathcal{K}$,*
(1) $\boldsymbol{VarSpec}_{\mathcal{K}}(q) \supseteq VSpec_{\mathcal{K}}(q)$,
(2) $\boldsymbol{VarSpec}_{\mathcal{K}}(q) \subseteq Spec_{\mathcal{K}}(q)$.

Proof. Claim (1):
Since we are replacing a variable in all possible ways while satisfying some conditions, it is sufficient to prove that the replacements which do not follow the conditions will not generate variable specializations.

if $x \in DistVar(q)$:
Condition 1: If we replace all the occurrences of a distinguished variable, the query is no longer valid as a result (distinguished) variable doesn't occur in the body of the query. So, we can't consider these replacements.
Condition 2: Since we are replacing x at one of its occurrences, the number of restrictions on the variable x are reduced. Thus we could obtain a specialization only if the possible values of x after the replacement, $P(x)'$, are less than the possible values of x before the replacement,$P(x)$. Hence, all those replacements which do not the follow this condition cannot be a specialization of q w.r.t. $\mathcal{K}$. The sub-condition $\exists u_i \in connected(x) : depends(u_i, y)$ identifies the replacements which aren't specializations directly without any membership checks.

if $x \in NonDistVar(q)$:
Condition 1: If the single occurrence of a variable is replaced then the number of variables in the new query are not equal to $|Var(q)|$. Thus, we do not obtain a specialization as condition 4 in Definition 3.1 is violated.
Condition 2: We can argue similar to the above *condition* 2 and show that when

the condition is not satisfied, the replacement will not produce any specialization.
Condition 3: It is essential for obtaining a specialization.

Hence, by the replacements in $varSpec_{\mathcal{K}}(q,x,y)$ we can generate all the variable specializations of q w.r.t. $\mathcal{K}$.

Claim (2): Proof by contradiction.
Assume, $\exists$ a query $q_1 \in \mathbf{VarSpec}_{\mathcal{K}}(q)$ and $q_1 \notin Spec_{\mathcal{K}}(q)$.
Then q_1 will not satisfy atleast one of the conditions in Definition 3.1.
Condition 1: All the replacements in $varRepl_{\mathcal{K}}(q,x,y)$ only generate queries which are strictly contained in q. If $x \in DistVar(q)$ we check for the condition $P(x) \supset P(x)'$ which ensures that $q' \subset_{\mathcal{K}} q$. If $x \in NonDist(q)$ we check the condition for query containment directly. So, condition 1 is always satisfied.
Condition 2: If $q' \in \mathbf{VarSpec}_{\mathcal{K}}(q)$ then $q'(\mathcal{K}) \neq \emptyset$ and the condition is directly satisfied.
Condition 3: Since we replace one occurrence of variable x with another variable y, where $x, y \in Var_o(q)$, we satisfy condition 3a which is sufficient to satisfy condition 3.
Condition 4: Since we use only the variables occurring atleast twice in the query for our replacements, condition 4 is always satisfied.

Hence, each query in $\mathbf{VarSpec}_{\mathcal{K}}(q)$ is a specialization of q w.r.t. $\mathcal{K}$. □

The minimal variable specializations of a query q w.r.t. $\mathcal{K}$ is defined by the set $\mathbf{minVarSpec}_{\mathcal{K}}(q) = \{q' \in \mathbf{VarSpec}_{\mathcal{K}}(q) \mid \exists$ no other query $q'' \in \mathbf{VarSpec}_{\mathcal{K}}(q)$ such that $q(\mathcal{K}) \subset q'(\mathcal{K})\ \}$.

3.2.6 Minimal variable individual specialization

Replacing a variable from a query with an individual can also produce a specialization of the query. For a variable $x \in Var_o(q)$ we need to replace x with an individual a at only one of its occurrences.

Definition 3.9. *A specialization q' of a query q w.r.t. $\mathcal{K}= (\mathcal{T}, \mathcal{R}, \mathcal{A})$ is called a* variable individual specialization (varIndv specialization) *of q w.r.t. $\mathcal{K}$ if q' is obtained from q by replacing a variable x with an individual a, where $x \in Var_o(q)$, $a \in Indv(\mathcal{A})$. A varIndv specialization q' of a query q w.r.t. $\mathcal{K}$ is called* minimal varIndv specialization *if $\exists$ no other varIndv specialization q'' such that $q' \subset_{\mathcal{K}} q'' \subset_{\mathcal{K}} q$.*

The set of all minimal variable individual specializations of q w.r.t. $\mathcal{K}$ is defined by $minVIndvSpec_{\mathcal{K}}(q)$ and $varIRepl_{\mathcal{K}}(q,x)$ defines the set of replacements of variable x in q with an individual $a \in Indv(\mathcal{A})$ which generate a minimal varIndv specialization of q w.r.t. $\mathcal{K}$.

Example 16. *Let a KB $\mathcal{K}$ contains the following assertions only*
$\mathcal{A} = \{$ *Man(a). Man(b). Man(c). Woman(d). Woman(e).*

isBrother(x, y) ≡ isBrother(y, x). isSister(x, y) ≡ isSister(y, x).
isBrother(a, b). isBrother(b, c). isBrother(c,a). isSister(d, e). isSister(e,d).}
Let, q(x) ← ∃y∃z.Man(x) ∧ isBrother(x, y) ∧ ⊤(y)
q_1(x) ← ∃y.Man(x) ∧ isBrother(x, b) ∧ ⊤(y) be two CQs. Here we can see that the result set of q(K) is {a, b, c} but for q_1(K) its just {a, c}. Thus q_1 is a varIndv specialization of q. But replacing z with d or e will not produce a specialization of q.

It is not an efficient way to generate the specializations by replacing a variable with a random individual in the ABox (and then checking if the new query is a specialization or not), as most of them don't generate specializations as shown in Example 16. So, we should avoid checking as many individuals as possible while computing the minimal varIndv specializations.

Computing Variable Individual Specializations

Let $varIndvRepl_{\mathcal{K}}(q,x)$ contain the replacements (of variable x in q with an individual and) which satisfy the below conditions

- none of the variables in the concept terms are replaced.
  - As after the replacement of x in a concept term $C(x)$, the term $C(a)$ either has value 0 or value 1 which makes the new query a non-specialization.
- when the variable x in the role term r(v, u) is being replaced and x is at the predecessor (or successor) position in the term $r(v, u)$, i.e. x $= v$ (or u), we replace x with the first descendant $d \in P(x)$ (Possible values of x as defined in Section 3.2.6) of $\top_i$ from each branch in the DG $\mathcal{G}_r^p$ ($\mathcal{G}_r^s$) such that the new query generated, q', has a reduced result set, $q'(\mathcal{K}) \neq q(\mathcal{K})$.
  - Higher level descendants of $\top_i$ contains the larger subset of successor/predecessor values.

We can generate all the minimal varIndv specializations by performing only the replacements in $varIndvRepl_{\mathcal{K}}(q,x)$, for each $x \in Var_o(q)$.
minVarIndvSpec$_{\mathcal{K}}(q) = \{q_1 \mid q_1(\mathcal{K}) \neq \emptyset$, where q_1 is obtained by the replacement $r \in varIndvRepl_{\mathcal{K}}(q,x)$, $\forall$x $\in Var_o(q)\}$.

Lemma 3.5. *Given a query q and a KB K*
(1) ***minVarIndvSpec***$_{\mathcal{K}}$*(q)* $\supseteq minVIndvSpec_{\mathcal{K}}(q)$,
(2) ***minVarIndvSpec***$_{\mathcal{K}}$*(q)* $\subseteq Spec_{\mathcal{K}}(q)$.

Proof. Claim (1):
It is sufficient to prove that the replacements in $varIndvSpec_{\mathcal{K}}(q,x)$ contains all minimal varIndv specializations.
i.e. replacement $r \in VarIRepl_{\mathcal{K}}(q, x) \rightarrow r \in varIndvRepl_{\mathcal{K}}(q, x,)$.
It is also sufficient to prove that all the replacements which do not follow the given conditions will not generate minimal variable specializations.
i.e. for all replacements $r \notin VarIndvRepl_{\mathcal{K}}(q, x) \rightarrow r \notin varIRepl_{\mathcal{K}}(q, x)$.

Condition 1: None of the variables occurring in a concept term are replaced by an individual.

Replacing x in concept term $C(x)$ with an individual a will not produce even a specialization because after modifying the term $C(x)$ into $C(a)$ either $\mathcal{K} \models C(a)$ or $\mathcal{K} \not\models C(a)$.

- $\mathcal{K} \models C(a) \Rightarrow$ new query is not a specialization (same set of results).
- $\mathcal{K} \not\models \mathrm{C}(a) \Rightarrow$ the result set of the new query is zero. So it is also not a specialization.

Condition 2: If we are replacing x in the term $r(x, z)$ or $r(z, x)$, we just need to show that (a) replacing x with a as mentioned in the conditions will always allow the biggest proper subset of initial possible values for z in the ABox $\mathcal{A}$, where z is the successor or predecessor of x in the role term where x is being replaced, and (b) the query q' obtained after the replacement has a reduced result set i.e. $q'(\mathcal{K}) \neq q(\mathcal{K})$.
In the directed graph $\mathcal{G}_r^p$ (or $\mathcal{G}_r^s$) individuals are ordered based on set inclusion. So, the higher level successors of the top individual $\top_i$ will always allow bigger proper subset of the possible values for z. Then, none of the remaining unselected individuals would provide z with more values than the ones at the topmost level and also generate a specialization. So, all replacements which do not follow condition 2 will not generate a minimal varIndv specializations.

Hence, the replacements in $varIndvRepl_{\mathcal{K}}(q,x)$ $\forall x \in Var_o(q)$ generate all minimal varIndv specializations of q w.r.t. $\mathcal{K}$.

Claim (2): Proof by contradiction.
Assume, $\exists$ a query $q_1 \in \mathbf{minVarIndvSpec}_{\mathcal{K}}(q)$ and $q_1 \notin Spec_{\mathcal{K}}(q)$.
Then q_1 will not satisfy atleast one of the conditions in Definition 3.1.
Condition 1: All the replacements in $varIndvRepl_{\mathcal{K}}(q, x)$ only generate queries which are strictly contained in q. So, condition 1 is always satisfied.
Condition 2: If $q' \in \mathbf{minVarIndvSpec}_{\mathcal{K}}(q)$ then $q'(\mathcal{K}) \neq \emptyset$ and the condition is directly satisfied.
Condition 3: Since we replace a variable x with an individual $a \in Indv(\mathcal{A})$, we always satisfy condition 3a which is sufficient to satisfy condition 3.
Condition 4: Since we convert our queries into CQNF, each variable occurs in atleast one concept term and this occurrence is not replaced. So, the number of variables remain unchanged and condition 4 is always satisfied.

Hence, each query in $\mathbf{minVarIndvSpec}_{\mathcal{K}}(q)$ is a specialization of q w.r.t. $\mathcal{K}$. □

3.2.7 Minimal individual specialization

Specializing a query by replacing an individual with another individual can produce specializations which are not covered by the earlier sections. But replacing with any random individual will not generate a specialization in most of the

cases. Nonetheless, we could obtain specializations in some of the cases like in Example 17 and they are called *individual specializations*.

Example 17. *Consider an ABox containing the following assertions.*
$\mathcal{A}$ = { *Man(d), Man(e), hasParent(d,a), hasParent(e,a), hasParent(d,b)* }
Among the CQs: $q(x) \leftarrow Man(x) \wedge hasParent(x, a)$
$q_1(x) \leftarrow Man(x) \wedge hasParent(x, b)$
we can see that $q_1(x)$ *is a (individual) specialization of* q*. The results of* $q_1(x)$ *is 1 while that of* $q(x)$ *is 2.*

Let, $Indv(q)$ represent all the individuals occurring in a query. Then an individual specialization and a minimal individual specialization of a query w.r.t. a KB are defined as follows.

Definition 3.10. *A specialization* q' *of* q *w.r.t.* $\mathcal{K}$= $(\mathcal{T}, \mathcal{R}, \mathcal{A})$ *is called an* individual specialization *of* q *w.r.t.* $\mathcal{K}$ *if* q' *is obtained from* q *by replacing an individual* a *with another individual* b*, where* $a \in Indv(q)$*,* $b \in \mathcal{A}$*. An individual specialization* q' *of a query* q *w.r.t.* $\mathcal{K}$ *is called* minimal individual specialization *if* $\exists$ *no individual specialization* q'' *of* q *w.r.t.* $\mathcal{K}$ *such that* $q' \subset_{\mathcal{K}} q'' \subset_{\mathcal{K}} q$*.*

The set of minimal individual specializations is defined by $minISpec_{\mathcal{K}}(q)$. Like in variable (and varIndv) specialization, we need to replace just one of the occurrences of the individual. We define $indvR_{\mathcal{K}}(q,a)$ as the set of replacements of an individual a in q with another individual which will generate a minimal individual specializations.

Computing Minimal Individual Specializations

Let $indvRepl_{\mathcal{K}}(q,a)$ contain the replacements satisfying the following two conditions.

- None of the individuals in concept terms are replaced.
- when the individual a in a role term r is being replaced and a is the predecessor (successor) of r, we replace a with the first descendant d of a from each branch in the sub-DG of $\mathcal{G}_r^p$ ($\mathcal{G}_r^s$) rooted at a such that the new query generated, q', has a reduced result set i.e. $q'(\mathcal{K}) \neq q(\mathcal{K})$.

We can generate all the minimal individual specializations by performing only the replacements in $indvRepl_{\mathcal{K}}(q,a)$, $\forall\ a \in Indv(q)$.
minIndvSpec$_{\mathcal{K}}(q) = \{q_1 \mid q_1(\mathcal{K}) \neq \emptyset$, where q_1 is obtained by the replacement $r \in indvRepl_{\mathcal{K}}(q,a)$, $\forall\ a \in Indv(q)\}$

Lemma 3.6. *Given a query* q *and a KB* $\mathcal{K}$= $(\mathcal{T},\mathcal{K},\mathcal{A})$
(1) $\boldsymbol{minIndvSpec}_{\mathcal{K}}(q) \supseteq minISpec_{\mathcal{K}}(q)$*,*
(2) $\boldsymbol{minIndvSpec}_{\mathcal{K}}(q) \subseteq Spec_{\mathcal{K}}(q)$*.*

Proof. Claim (1):
It is sufficient to prove that the replacements in $varIndvSpec_{\mathcal{K}}(q,a)$ generates

all minimal individual specializations.

i.e. replacement $r \in indvR_{\mathcal{K}}(q, a) \rightarrow r \in indvRepl_{\mathcal{K}}(q, a)$

It is also sufficient to prove that all the replacements which do not follow the given conditions will not generate minimal individual specializations.

i.e. $r \notin indvRepl_{\mathcal{K}}(q, a) \rightarrow r \notin indvR_{\mathcal{K}}(q, a)$

Condition 1: None of the individuals occurring in a concept term are replaced.

Replacing a in concept term $C(a)$ with another individual b will not produce even a specialization because after rewriting the term $C(a)$ into $C(b)$, either $\mathcal{K} \models C(b)$ or $\mathcal{K} \not\models C(b)$.

- $\mathcal{K} \models C(b) \Rightarrow$ new query is not a specialization (same set of results).
- $\mathcal{K} \not\models \mathrm{C}(b) \Rightarrow$ the result set of the new query is zero, not a specialization.

Condition 2: We just need to show that (a) replacing a with b as above will always allow the biggest proper subset of the possible values for z in the ABox $\mathcal{A}$, where z is the successor or predecessor of a in the role term where a is being replaced, and (b) the query q' obtained after the replacement has a reduced result set i.e. $q'(\mathcal{K}) \neq q(\mathcal{K})$.

In the directed graph $\mathcal{G}_r^p$ (or $\mathcal{G}_r^s$) individuals are ordered based on set inclusion. So, the higher level descendants of the individual a in the DGs will always allow the bigger proper subset of the possible values for z. So, none of the remaining individuals would provide z with more values and also generate a specialization. This implies, all replacements which do not follow condition 2 will not generate a minimal individual specialization.

Hence, the replacements in $indvRepl_{\mathcal{K}}(q,a)$, $\forall a \in Indv(q)$ generate all minimal individual specializations of q w.r.t. $\mathcal{K}$.

Claim (2): Proof by contradiction.
Assume, $\exists$ a query $q_1 \in \mathbf{minIndvSpec}_{\mathcal{K}}(q)$ and $q_1 \notin Spec_{\mathcal{K}}(q)$.
Then q_1 will not satisfy atleast one of the conditions in Definition 3.1.
Condition 1: All the replacements in $indvRepl_{\mathcal{K}}(q, a)$ only generate queries which are strictly contained in q. So, condition 1 is always satisfied.
Condition 2: If $q' \in \mathbf{minIndvSpec}_{\mathcal{K}}(q)$ then $q'(\mathcal{K}) \neq \emptyset$ and the condition is directly satisfied.
Condition 3: Since we replace an individual a with an individual b, where $a, b \in Indv(\mathcal{A})$, we satisfy condition 3a which is sufficient to satisfy condition 3.
Condition 4: Since we replace only the individuals in the query, the variables in the query remain unchanged and condition 4 is always satisfied.

Hence, each query in $\mathbf{minVarIndvSpec}_{\mathcal{K}}(q)$ is a specialization of q w.r.t. $\mathcal{K}$. □

3.2.8 Post-processing steps

The queries generated in the previous steps contain concept and role names introduced while normalizing the KB along with variables introduced while con-

verting the original query into its CQNF. In addition, we can eliminate the nominals occurring in the specializations to reduce the number of variables in the specializations to a minimum. Hence, the following three post-processing steps can be performed recursively before presenting the computed minimal specializations to the user.

1. Remove the concept and role names introduced while converting the KB into its normal form.
   (a) For every new concept name C introduced $\exists$ a concept inclusion $C_1 \sqcap C_2 \sqsubseteq C$, and each occurrence of C is replaced by the $\mathcal{EL}^{++}$-concept $C_1 \sqcap C_2$.
   (b) For every new role names r introduced $\exists$ a role inclusion $r_1 \circ r_2 \sqsubseteq r$, and each occurrence of r is replaced by the $\mathcal{EL}^{++}$-role $r_1 \circ r_2$.
2. Eliminate the new variables introduced during the conversion of the query to CQNF. We perform this using the following rules.
   - If $\exists r.C(x) \rightsquigarrow r(x,y) \wedge C(y)$ and
     - r is specialized to r' then, $r'(x,y) \wedge C(y) \rightsquigarrow \exists r'.C(x)$
     - C is specialized to C' then, $r(x,y) \wedge C'(y) \rightsquigarrow \exists r.C'(x)$
   - $r_1 \circ r_2(x,y) \rightsquigarrow r_1(x,z) \wedge r_2(z,y)$ and
     - r_1 is specialized to r'_1 then, $r'_1(x,z) \wedge r_2(z,y) \rightsquigarrow r'_1 \circ r_2(x,y)$
     - r_2 is specialized to r'_2 then, $r_1(x,z) \wedge r_2(z,y) \rightsquigarrow r_1 \circ r'_2(x,y)$
     - role term $r'_1(x,z)$ is added then, $r_1(x,z) \wedge r_2(z,y) \wedge r'_1(x,z) \rightsquigarrow r_1 \circ r_2(x,y) \wedge r'_1 \circ r_2(x,y)$
     - role term $r'_2(z,y)$ is added then, $r_1(x,z) \wedge r_2(z,y) \wedge r'_2(z,y) \rightsquigarrow r_1 \circ r_2(x,y) \wedge r_1 \circ r'_2(x,y)$
3. Eliminate the nominals in the query, as performed in the conversion steps of CQNF.

The above post-processing steps also makes the specializations concise increasing their understandability. The function which recursively performs the above 3 steps until no further reduction is possible is denoted by Denormalize$_{\mathcal{K}}(q)$. It takes a set of queries as input and generates a set of equivalent (de-normalized) queries.

3.3 Algorithm: *MinSpec*

We can generate all the minimal specializations of a query q w.r.t. $\mathcal{K}$ by using the above six methods introduced in the Sections 3.2.2 - 3.2.7. The set of all minimal specializations of q are contained in T where
T := $\mathbf{minIndvSpec}_{\mathcal{K}}(q) \cup \mathbf{minVarIndvSpec}_{\mathcal{K}}(q) \cup \mathbf{minVarSpec}_{\mathcal{K}}(q) \cup \mathbf{minConjSpec}_{\mathcal{K}}(q) \cup \mathbf{minRoleSpec}_{\mathcal{K}}(q) \cup \mathbf{minConcSpec}_{\mathcal{K}}(q)$.

T contains all the minimal specializations of q, but not all the queries in T are minimal specializations of q. For example, a varIndv specialization q_1 can be contained in a concept specialization q_2 and vice-versa. So, we need to extract the minimal specializations,$minSpec_{\mathcal{K}}(q)$, from the specializations computed from the above methods. The following algorithm, $MinSpec$, calculates the set of minimal specializations of a CQ q w.r.t. a KB $\mathcal{K}$. It takes a conjunctive query q as input and we assume that the pre-processing steps described in Section 3.2.1 are performed. A set of minimal specializations of q w.r.t. $\mathcal{K}$ are provided as an output.

Algorithm *MinSpec*
Input: $\mathcal{EL}^{++}$-CQ q, $\mathcal{EL}^{++}$-KB $\mathcal{K}$
Output: Q - set of all minimal specializations of q w.r.t. $\mathcal{K}$
if q has an empty result set
then return Q
$q' \leftarrow NORMALIZE_{\mathcal{K}}(q)$
T $\leftarrow$ **minIndvSpec**$_{\mathcal{K}}(q')$ $\cup$ **minVarIndvSpec**$_{\mathcal{K}}(q')$ $\cup$ **minVarSpec**$_{\mathcal{K}}(q')$ $\cup$ **minConcSpec**$_{\mathcal{K}}(q')$ **minConjSpec**$_{\mathcal{K}}(q')$ $\cup$ **minRoleSpec**$_{\mathcal{K}}(q')$.
R $\leftarrow \{q_1 \in \text{T} \mid \exists \text{ no } q_2 \in \text{T} \text{ such that } q_1 \subset_{\mathcal{K}} q_2\}$.
Q $\leftarrow Denormalize_{\mathcal{K}}(\text{R})$
return Q

Q is the set of all minimal specializations of q w.r.t. $\mathcal{K}$.

Now, we present the results of soundness, completeness and termination of the above proposed algorithm.

3.3.1 Soundness and Completeness

Showing the soundness of the algorithm *MinSpec* is trivial once we proved the completeness as we check the containment of a query $q_2 \in$ T in another query $q_1 \in$ T in our algorithm.

Theorem 3.1. *Given an $\mathcal{EL}^{++}$-conjunctive query q and an $\mathcal{EL}^{++}$-KB $\mathcal{K}$, a query $q' \in MinSpec_{\mathcal{K}}(q)$ if and only if q' is a minimal specialization of qw.r.t. $\mathcal{K}$.*

Proof. COMPLETENESS:
q' is a minimal specialization of q w.r.t. $\mathcal{K} \implies q' \in MinSpec_{\mathcal{K}}(q)$.
To obtain a minimal specialization, we need to specialize a query only once using any one of the 6 methods. From the Lemma 3.1 - Lemma 3.6 it directly follows that T contains all the minimal specializations of q w.r.t. $\mathcal{K}$. Since we do not delete any minimal specializations of q w.r.t. $\mathcal{K}$ while generating Q, all the minimal specializations of q w.r.t. $\mathcal{K}$ occur in Q.
Hence, q' is a minimal specialization of q w.r.t. $\mathcal{K} \implies q' \in MinSpec_{\mathcal{K}}(q)$.

SOUNDNESS: $q' \in \mathit{MinSpec}_{\mathcal{K}}(q) \implies q'$ is a minimal specialization of q.

This is a trivial proof as in our algorithm *MinSpec* we check and remove the non-minimal specializations. So, we prove by contradiction.
Assume that there is query $q_1 \in \mathit{MinSpec}_{\mathcal{K}}(q)$ which is not a minimal specialization of q w.r.t. $\mathcal{K}$. Then this query q_1 is contained in another query q_2 which is a minimal specialization. But during our algorithm execution the query q_1 will be removed as it is contained in q_2. So, $q_1 \notin MinSpec_{\mathcal{K}}(q)$, a Contradiction.
Hence, q' is a minimal specialization of q w.r.t. $\mathcal{K} \impliedby q' \in \mathit{MinSpec}_{\mathcal{K}}(q)$. □

3.3.2 Termination and Complexity

The maximum number of specializations obtained by $\mathbf{minConcSpec}_{\mathcal{K}}(q)$, $\mathbf{minRoleSpec}_{\mathcal{K}}(q)$and $\mathbf{minConjSpec}_{\mathcal{K}}(q)$ is bounded by the number of concept names and role names existing in TBox $\mathcal{T}$, number of terms in the query, and these values are finite. The maximum number of specializations obtained by $\mathbf{minVarSpec}_{\mathcal{K}}(q)$, $\mathbf{minVarIndvSpec}_{\mathcal{K}}(q)$ and $\mathbf{minIndvSpec}_{\mathcal{K}}(q)$ is bounded by the number of variables, individuals and their occurrences in q and also by the size of the ABox. So, the number of specializations in T is finite. Calculating Q, the set of minimal specializations, takes at most $|\mathrm{T}|^2$ steps. Thus we can see that the algorithm *MinSpec* has only finitely many steps which proves that it will always terminate.

While calculating $DS^q_{\mathcal{K}}(C)$ for a concept C w.r.t. $\mathcal{K}$, we perform o(|roles| × |concept names|) number of subsumption and containment checks to see if a concept in $ERC_{\mathcal{K}} \cup NC_{\mathcal{K}}$ is directly subsumed. Similarly, while calculating $NC^q_{\mathcal{K}}(C)$ we perform o(2 × (|roles| * |concept names| + |concept names|)) number of subsumption and containment checks. |roles| * |concept names| corresponds to the concepts in $ERC_{\mathcal{K}}$ and |concept names| corresponds to concepts names which are not comparable to C w.r.t. $\mathcal{K}$. To calculate $DNS^q_{\mathcal{K}}(C)$, we perform $|DS^q_{\mathcal{K}}(C)|$ * $|\mathcal{A}|$ number of membership checks. So the number of subsumption, containment and membership checks required to calculate $\mathit{minConcSpec}_{\mathcal{K}}(C)$ is o($|\mathrm{KB}|^2$ + |KB|), o($|\mathcal{K}|$*$|\mathcal{A}|$). Moreover, to obtain all the minimal concept specializations, we have to replace each k conjuncts in n concepts terms in the query. Let |Q| represent the size of the query. Then, the total number of (subsumption, containment and membership) checks required to calculate $\mathbf{minConcSpec}_{\mathcal{K}}(q)$ is o($|\mathrm{Q}|^2$ × | KB $|^2$).

Similarly to calculate $\mathbf{minRoleSpec}_{\mathcal{K}}(q)$ requires o($|\mathcal{R}|$ * $|\mathcal{R}|$ * | Q |) number of sumbsumption and containment checks. To calculate $\mathit{InCompRole}_{\mathcal{K}}(r)$ we require o($|\mathcal{R}|^2$) number of subsumption and containment checks. To calculate $\mathbf{minConjSpec}_{\mathcal{K}}(q)$ we require additional o($|\mathcal{R}|^2$) number of subsumption and containment checks. Let n be the number of variable in a query and k be the number of occurrences of a variable. Then we perform o(n*k*(n-1)) number of replacements for computing minimum variable specializations. Calculating $\mathit{depends}(x, y)$ and $\mathit{connected}(x, y)$ will require |Q|, $|\mathrm{Q}|^2$ number of steps

respectively. Checking the conditions requires $|\mathrm{Q}|^3 + |\mathrm{Q}|^2$ number of containment checks. The containment checks and replacement required for calculating $\mathbf{minVarSpec}_{\mathcal{K}}(q)$ is $\mathrm{o}(|\mathrm{Q}|^3)$. To calculate $\mathbf{minVarIndvSpec}_{\mathcal{K}}(q)$ we perform $\mathrm{o}(|\mathrm{Q}|^2 \times |\mathrm{KB}|)$ number of replacements and containment checks. Here, $|\mathrm{KB}|$ is from the number of individuals with which we replace a variable (size of ABox). Similarly, the number of replacement and containment checks required for calculating $\mathbf{minIndvSpec}_{\mathcal{K}}(q)$ is $\mathrm{o}(|\mathrm{Q}|^2 \times |\mathrm{KB}|)$. $\mathrm{Denormalize}_{\mathcal{K}}(q)$ is also be performed in PTIME.

We already know that the complexity of subsumption checking in regular $\mathcal{EL}^{++}$ is PTIME and the complexity of CQ containment checking in regular $\mathcal{EL}^{++}$ is NPTIME. So the combined complexity of the algorithm *MinSpec* has an upper bound of NPTIME.

Chapter 4

Clustering of Query Results

The method $MinSpec$ generates the set of all minimal specializations of a query q w.r.t. a KB $\mathcal{K}$. Each minimal specialization generated reduces the information overload of the query results. To reduce information overload by a significant amount, further manual selection (i.e. selecting the union or intersection of some of the minimal specializations) is required. But, this manual selection is not a trivial task. So, we developed an alternative method which reduces the information overload significantly and at the same time simplifies the selection process. This approach is based on clustering of the query result tuples i.e. grouping of similar result tuples and it *specializes a conjunctive query q in $\mathcal{EL}^{++}$ w.r.t. an $\mathcal{EL}^{++}$-KB $\mathcal{K}$ by clustering the result tuples of the query result $q(\mathcal{K})$*. Unlike $MinSpec$, this approach do not perform query containment checks making it more time-efficient.

Using the concept and role hierarchies obtained from the KB, we can easily construct a tree (resembling a hierarchical clustering) based on the concepts and roles appearing in the query. Using this tree, we can group the result tuples to form clusters. Corresponding to each of these cluster we can also generate a query using the cluster's (tree node's) label. In Section 4.1, we introduce the basic concepts of clustering and various types of clustering techniques. We also introduce some of the important subtasks related to hierarchical clustering. In Section 4.2, we explain how clustering can be used to specialize a query. We also present the drawbacks of the previous clustering based approach(es) and illustrate the improvement required to overcome those shortcomings. In Section 4.3, we present the important steps of our method to generate clusters and their corresponding queries. In Section 4.4, we present our final algorithm along with complexity results.

4.1 Clustering

"Clustering (or *cluster analysis*) aims to organize a collection of data items (in our case result tuples) into clusters, such that items within a cluster are more 'similar' to each other than they are to items in the other clusters" [Grira *et al.*, 2004]. A *cluster* is a group of the same or similar elements gathered or occurring closely together [TheFreeDictionary, 2011]. Clusters are implicit groupings in data items and helps the user understand these groups by providing labels. When provided with appropriate labelling, the user can easily choose one or more clusters and eliminate the irrelevant data items reducing the information overload significantly.

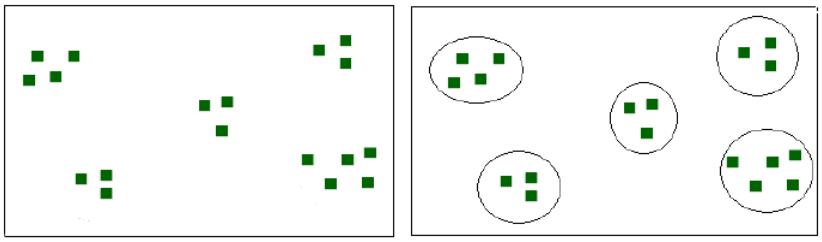

Figure 4.1: Partitional Clustering

There are two types of clustering structures: *partitional* and *hierarchical*. *Partitional clustering* groups the data items into non-overlapping subsets such that each result tuple belongs to just one cluster (see Figure 4.1). The number of clusters and partitioning criterion is pre-decided. Examples of partitional clustering algorithms are k-means and k-medoids.

Hierarchical clustering groups data items into overlapping subsets (resembling a tree structure) where each data item is in (one or) many clusters (see Figure 4.2). *Hierarchical clustering* can be performed in two ways: *divisive clustering* and *agglomerative clustering*. *Divisive clustering* initially assigns all data items to one cluster. The cluster is then divided until each result tuple is in a cluster by itself thereby creating a (nested) hierarchical clustering. *Agglomerative clustering* begins with each result tuple in its own cluster, and then combines two (or more) clusters based on some similarity measure until all result tuples are in one cluster.

At any point in the hierarchical structure a proximity bound can be imposed creating a partitional clustering of the data. This is known as *Partitional Clustering from a Hierarchical Clustering* (see Figure 4.3). A partitional clustering can be generated from a hierarchical clustering by cutting the tree, and if the hierarchical clustering is agglomerative it is also known as *constrained hierarchical clustering*. For divisive clustering, we call it *constrained divisive clustering*.

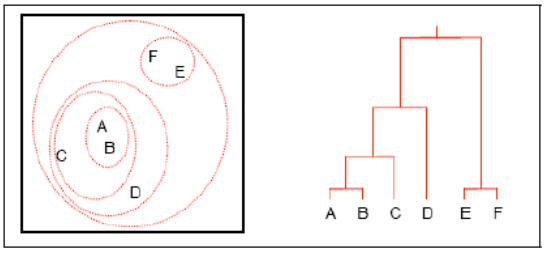

Figure 4.2: Hierarchical Clustering

For our purpose, we extend constrained divisive clustering by assigning non-leaf nodes with tuples which are not assigned to any of their descendants and we call this structure *contrained divisive hierarchical clustering* (CDHC). Constrained Divisive Hierarchical Clustering (CDHC) is the most appropriate clustering structure in our case, because

1. Using the concept and role refinement operators, we can easily construct a tree (with a top-down approach) based on some features extracted from the query and the KB.
2. Each tuples can belong to more than one cluster providing a natural hierarchy among clusters.
3. We need to restrict the number of clusters to be provided to the user.
4. An individual can be a member of a concept C and none of its minimal specializations w.r.t. the KB. Similarly for a role r the pair of individuals (a, b) may not be a member of any of its minimal specializations w.r.t. the KB.

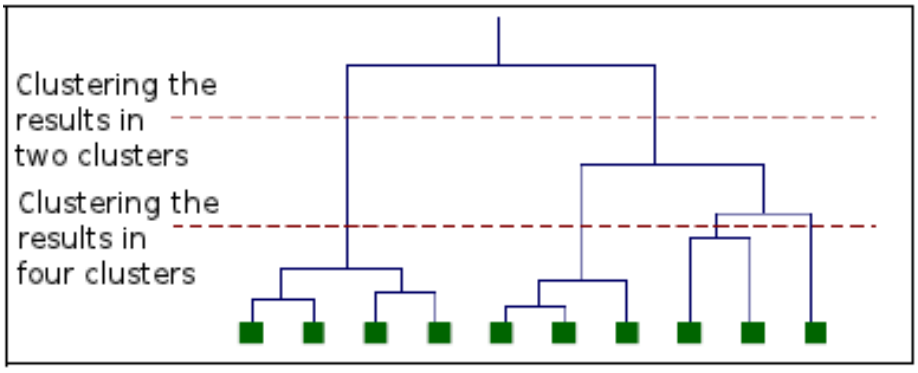

Figure 4.3: Partitional Clustering from a Hierarchical Clustering

There are few important subtasks in every divisive hierarchical clustering algorithm. We discuss here some of the sub-tasks which are relevant for our approach.

4.1.1 Feature Selection

Feature selection is a process of selecting a subset of available features using some criteria [Zhao *et al.*, 2010]. It aims at ignoring the features which are most irrelevant and unnecessary based on certain evaluation criteria. By this, (1) the speed of learning process and model interpretability are increased, and (2) the problem of dimensionality is reduced. A typical feature selection algorithm contains 4 main steps as shown in Figure 4.4 namely, *subset creation*, *subset evaluation*, *stopping criterion* and *result validation*.

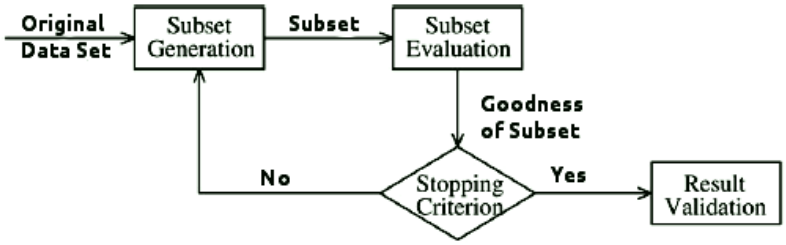

Figure 4.4: Four key steps of Feature Selection

Feature selection algorithms developed using various strategies fall in one of the following three categories: **filter, wrapper and embedded(hybrid)** models. The *filter* model algorithms depend on data's characteristics alone and selects the features without using any mining or learning algorithm. The *wrapper* model algorithms use pre-determined learning algorithms and select the features based on the features' performance. Algorithms in *hybrid* model aim at combining advantages of both models while overcoming some of the disadvantages. In our case, feature selection algorithms belonging to the filter category are appropriate, as we can select the features based on the tuples and the KB alone without using a learning process.

4.1.2 Stopping Criteria

In divisive hierarchical clustering, the number of clusters is not specified a priori and the clustering continues until each cluster contains only one tuple (data item). However, for CDHC it is clear that the clustering algorithm should be stopped much before each tuple is placed in a separate cluster. So, a stopping criteria can be computed based on which the splitting of the nodes is stopped. Two ways to compute the stopping criteria are,(1) by selecting the number of

clusters to be generated, **CL**, and (2) by selecting the maximum number of tuples that a cluster can contain, **S**. From one of these two values, an approximate value for the other can be derived easily using the formula $\mathbf{S} = \frac{|q(\mathcal{K})|*\beta}{\mathbf{CL}}$, where β is some constant and $\beta > 1$. In our algorithm, we request the user to provide one of these two values using which we determine the stopping criteria.

4.1.3 Cluster Selection for splitting

A divisive hierarchical clustering recursively splits a cluster into two or more clusters starting from the original set of result tuples. The selection of the clusters for splitting can have a remarkable impact on the overall clustering result. There are three approaches to perform the above task [Savaresi *et al.*, 2002],

- complete partition: every cluster is split
- cluster containing the maximum number of elements (tuples) is split
- the cluster with the highest variance with respect to its centroid is split.

In our method, the cluster containing the maximum number of tuples is split and after each split the fulfilment of the stopping criteria can be checked easily.

4.2 Specializing a query using clustering

Specializing a query q by clustering its results is fairly simple once we have an efficient clustering algorithm for the result tuples of a query over a DL based KB. Using the result tuples and the KB $\mathcal{K}$, a constrained divisive hierarchical clustering can be performed, generating a cluster tree. A *cluster tree* is the hierarchy of clusters obtained from a (constrained divisive) hierarchical clustering. The construction of the cluster tree is halted, when a stopping criteria is reached (like obtaining the required number of clusters, n). The nodes in the cluster tree (except the root node) assigned with atleast one tuple are treated as clusters. Depending on the KB, we may or may not generate clusters, and in addition, some of the result tuples can be assigned to the root node of the cluster tree resulting in the exclusion of some of the result tuples from the generated clusters. Formally, the problem of clustering the query results can be defined as follows:

Definition 4.1. *Given a query q and DL KB $\mathcal{K}$, a* clustering algorithm *for the result tuples of q w.r.t. $\mathcal{K}$ generates n clusters $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n$ where $n \geq 0$, $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \cdots \cup \mathcal{C}_n \subseteq q(\mathcal{K})$ and $\emptyset \subset \mathcal{C}_i \subset q(\mathcal{K})$, $\forall i \in [1...n]$.*

The number of clusters generated is approximately the number of clusters suggested by the user. From these n clusters, n new non-empty queries which are strictly contained in q can be generated (using the labelling of the clusters).

The n new queries generated may no longer be *specialization*s of the query q w.r.t. $\mathcal{K}$ as defined in Definition 3.1.

When it comes to result tuples clustering, we can make use of the knowledge provided in the knowledge base in terms of concept and role inclusions only. To use the ordering between individuals from the collection $\mathcal{F}$, we also need to have role names in the result tuples which is not possible. So, we need to alter the concepts and roles occurring in the query only.

For this purpose we introduce a new term, *c-specialization*. A *c-specialization* is obtained by changing multiple concepts and roles occurring in the query q as defined in Definition 4.2.

Definition 4.2. *A conjunctive query q' is called a* c-specialization *of a conjunctive query $q \leftarrow p_1 \wedge p_2 \wedge \cdots \wedge p_m$ w.r.t. $\mathcal{K}$ if*

1. *$q'(\mathcal{K}) \subset q(\mathcal{K})$*
2. *$q'(\mathcal{K}) \neq \emptyset$*
3. *$q' \leftarrow p'_1 \wedge p'_2 \wedge \cdots \wedge p'_m$, where either $p'_i = p_i$ or p'_i is obtained by replacing the concept or role in p_i.*

It is represented by $q' \Subset_{\mathcal{K}} q$.

Thus, our task is to generate n c-specializations of a query q w.r.t. $\mathcal{K}$, where n is approximately the number of clusters requested by the user. From these n c-specializations, the user can select $i \in [1 \ldots n]$ clusters to generate the required specialized result set and the corresponding union of conjunctive queries.

Previous approach(es) and its drawbacks

The method proposed here is based on the work done in [Amato *et al.*, 2010]. The general idea of [Amato *et al.*, 2010] is, instead of aggregating the results based on values in tuples, aggregation is done based on semantic information linked to the values in the tuples, i.e. concept and role (inclusions and) memberships inferred from the ontologies. The user first selects the *result variables* (i.e. distinguished variables) to be used for the clustering process. Corresponding to each selected result variable, a complex concept is extracted from the query. These complex concepts are called *types* of the result variables and can be considered as features of the clustering algorithm. Corresponding to each type, a tree can be constructed using the concept hierarchy generate from the KB. These trees are then combined using a binary tree-product operator to obtain a combined tree and by assigning each result tuple to one of the nodes in the combined tree , a set of clusters will be generated. Figure 4.5 gives an example combined tree obtained by clustering two trees rooted at types A, A′ selected for clustering. There are several drawbacks in the method proposed by [Amato *et al.*, 2010] w.r.t. an $\mathcal{EL}^{++}$-KB. Some of them are:

1. It argues that the trees corresponding to each type can be constructed using the concept hierarchy obtained from the KB. Since the type of a result variable can be a complex concept, we can't rely on the concept hierarchy alone and we need advanced concept refinement operators.

2. The role inclusions available in the RBox are not utilized. Thus, generating optimal clusters may not be possible. Especially, role inclusions of the form $r \circ s \sqsubseteq r$, $s \circ r \sqsubseteq r$ can easily generate a large cluster tree rooted at role r which can improve the quality of the clusters.

3. In an $\mathcal{EL}^{++}$-KB, we can't specify the range of a role directly. This is only a minor drawback in $\mathcal{EL}^{++}$-KBs, however in other DLs like OWL 2 EL they can be specified easily.

4. Selection of the results variables can be automated to reduce user dependency and improve the efficiency of the clustering. In many cases, the users are not experienced enough to select the optimal subset of the result variables which can result in the generation of non-optimal clusters as shown in Figure 4.5.

5. While generating the trees corresponding to each type, no stopping condition is provided. This results in the generation of more than the required number of clusters.

Improvements performed in our approach

Instead of relying on concept hierarchy, a concept refinement operator like $minConcSpec_{\mathcal{K}}(C)$ will be used to construct the trees corresponding to each type selected for clustering. For each result variable, in addition to a type which is a concept (called as *concept type* or *ctype*) we also extract ≥ 0 roles from the query and they are called as *role types* (or *rtypes*). Using these rtypes and role inclusions inferred from the KB, we will also construct trees rooted at these rtypes. Since we use both ctypes and rtypes, it is not necessary to have domain and range values of each role occurring in the query. Using various feature selection algorithms, we will directly select the set of *types* (ctypes and rtypes) to be used for clustering.

The user can suggest either the number of clusters to be generated or the maximum size of a cluster, by which he/she chooses the level of flexibility he/she needs to obtain the required specialization. This also solves the problem of too many clusters. Dividing the query results into 2 clusters provides fewer c-specializations when compared to dividing the query results into 10 clusters. From the 10 clusters, user can select any $9, 8, \ldots, 2$ or even 1 cluster as the specialized result set, giving him/her more options to specialize the query. Our method also employs optimization techniques (like integration of the steps, (a) tree construction, and (b) grouping of the result tuples in the cluster tree) and heuristics to efficiently generate optimal clusters of query results over a DL KB.

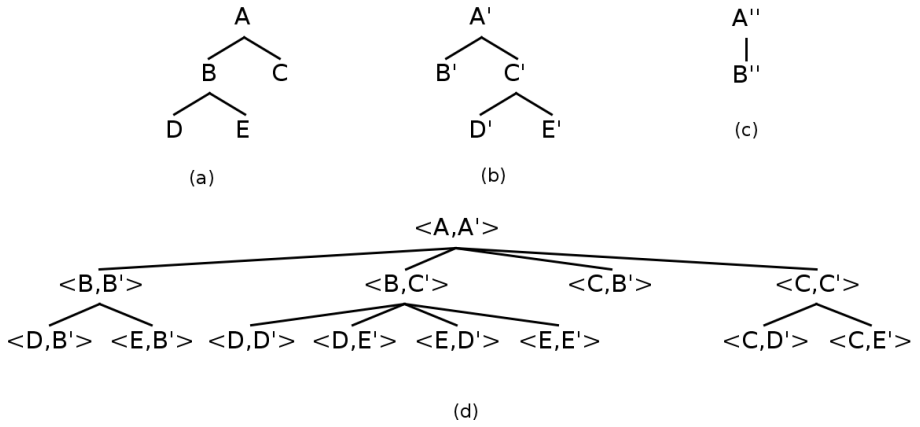

Figure 4.5: Generation of a cluster tree
Consider the query $q(x, y, z) \leftarrow C(x) \wedge C'(y) \wedge C''(z) \wedge r(x, z)$. Let A, A$'$, A$''$ be the types of x, y, z i.e. $C \sqcap Domain(r)$, C', $C'' \sqcap Range(r)$ respectively. (a), (b) and (c) are the two cluster trees rooted at A, A$'$ and A$''$ respectively. We cluster the result tuples using the types of the variables x, y. (d) is the combined cluster tree obtained by combining the trees in (a) and (b) using the tree-product operator. Using the tree from (c) to construct the combined tree can potentially reduce the number of result tuples assigned to the non-root nodes causing a reduction in the quality of the clusters. So, we avoid utilizing (c) to construct the combined cluster tree.

In addition, using the generated clusters and its labels, we provide the user a set of c-specializations (whose result set form each cluster). The user then selects only the relevant queries, removing the irrelevant result tuples, thus providing a specialization of the original query result. The corresponding query is the union of the queries selected by the user. When we say that a query or result tuple is irrelevant, we mean that the user is not interested in those result tuples.

4.3 Generating clusters and c-specializations

The following sections describe the main steps of our algorithm for specializing a query q over a DL KB $\mathcal{K}$ by clustering the query result tuples - $q(\mathcal{K})$. As in Chapter 3, we convert our queries into the normal form CQNF and in the post-processing steps all newly introduced variables are eliminated. We also assume that we compute and store the results of $minConcSpec_{\mathcal{K}}(C)$, $minRoleSpec_{\mathcal{K}}(r)$, $\forall C \in \mathrm{N}_C^{\mathcal{K}}, \forall r \in \mathrm{N}_R^{\mathcal{K}}$ where $\mathcal{K}$ is a normalized and regular $\mathcal{EL}^{++}$-KB. The functions $minConcSpec_{\mathcal{K}}(C)$, $minRoleSpec_{\mathcal{K}}(r)$ can be seen as concept and role refinement operators.

As we have seen earlier, feature selection is the process of selecting a subset of the available features and is one of the first steps in the clustering process. Before we perform the selection of the features, we first need to extract the features (also called as *types*) from the query.

4.3.1 Getting the types of a result variable

The values of a variable in a query are dependent on the concepts and roles occurring in the query and changing these concepts and roles alters the set of possible values of the variable. Thus, we can cluster the result tuples using the KB, concepts and roles corresponding to the result variables of the query and these concepts and roles are called *types*. There are two kinds of types, (1) *ctypes* (which are concepts), and (2) *rtypes* (which are roles). Since in $\mathcal{EL}^{++}$ the constructor $\sqcap$ can be used over concepts only, for each result variable has one concept type (ctype), and ≥ 0 role types (rtypes).

Let $\mathcal{B}_i$ be the ctype of the variable $v_i \in DistVar(q)$ and $\mathcal{B}_i$ is obtained by combining all the concepts from the concept terms containing v_i. Since the query under consideration is in CQNF, there exists only one concept term for each result variable. Thus, a ctype $\mathcal{B}_i$ of each variable $v_i \in DistVar(q)$ can be obtained directly from a concept term as shown in Example 18.

Example 18. *Given query is* $q(x, y) := C \sqcap E(x) \wedge D(y)$
The ctype of variable x *is* $C \sqcap E$.
The ctype of variable y *is* D.

Similarly, by using the role terms occurring in the query, rtypes for each variable $v_i \in DistVar(q)$ can be extracted. If $r(u, v_i)$ or $r(v_i, u) \in q$ and $u \in DistVar(q)$ then r is a role type of v_i. Since both v_i and u need to be in $DistVar(q)$, the number of rtypes obtained are lower than the number of role terms in q. However, there can be more than one rtypes for a variable $v_i \in DistVar(q)$ and one role r can be a rtype of more than one result variable.

Example 19. *Given a query*
$q(x, y) := C \sqcap E(x) \wedge D(y) \wedge r(x, y)$
The types obtained from variable x *are* $\{C \sqcap E, r\}$.
The types obtained from variable y *are* $\{D, r\}$.
Here, $Var(C, q) = x$, $Var(D, q) = y$ *and* $Var(r, q) = (x, y)$
For the query, $q_1(x) := C(x) \wedge D(y) \wedge E(z) \wedge r(x, y) \wedge s(y, z)$
The types obtained from variable x *are* $\{C\}$.
Since, $y \notin DistVar(q)$ *we can't obtain any type corresponding to* y.

As shown in Example 19, we can extract the types (both ctypes and rtypes) of all the result variables quite easily. We define the set of rtypes of a variable $v_i \in \mathtt{V}$ and q as

$$\mathbf{RTYPE}(v_i, q) := \{r \mid r(u, v_i) \vee r(v_i, u) \in q \wedge u \in DistVar(q)\}$$

We also define the following sets to define the set of all ctypes, rtypes and types obtained from the result variables of a query respectively.

$\mathbf{CTYPE}(q) := \{\mathcal{B}_i \mid \mathcal{B}_i \text{ is a ctype of } v_i \in DistVar(q)\}$
$\mathbf{RTYPE}(q) := \{\mathbf{RTYPE}(v_i, q) \mid v_i \in DistVar(q)\}$
$\mathbf{TYPE}(q) := \mathbf{CTYPE}(q) \cup \mathbf{RTYPE}(q)$

The function $\mathbf{POS}(t_i)$ returns the position of the term from which the type t_i is obtained. A ctype t_i is also represented as t_i^c and a rtype t_j is also represented as t_j^r. The variables corresponding to each type are stored using the function Var, where

$Var(C, q) = v_i$, where C is the ctype of variable v_i
$Var(r, q) = (v_i, v_j)$, where r is the rtype of either v_i or v_j, $r(v_i, v_j)$ is a term in q, and the other variable $\in DistVar(q)$.

The set $\mathbf{TYPE}(q)$ is the set of all features which can be used for clustering the query results. However, only some of these features are relevant for clustering and eliminating the unnecessary features can improve the performance of the clusters. As a next step, we perform the selection of a subset of features which will be used in our clustering algorithm.

4.3.2 Selecting the features for clustering

The task of selecting relevant features in clustering or classification problems can be viewed as one of the most fundamental problems in the field of machine learning. Corresponding to each result variable we obtain one or more *types*. One way to select the features is to ask the user directly and it is a normal practice in Database aggregate usage (like GROUP-BY) where user provides the input variables to perform the operation. This makes our algorithm highly supervised. Instead of obtaining the subset of features from the user or by (semi-)supervised algorithms, we can select them automatically. As discussed in Section 1.2, there are several unsupervised feature selection algorithms like [Dash and Liu, 2000], [Cai *et al.*, 2010] which can be applied to our query result tuples.

As we perform clustering using our new approach, algorithms with slightly higher complexity but efficient for further clustering need to be used. We now present the basic steps of such an algorithm which selects the subset of the types (features) obtained from the result variables of a query q. Corresponding to each of the n result variable in q, we extract the n ctypes. From these n ctypes, we generate n trees rooted at these n ctypes (as explained in Section 4.3.5). These n trees are then compared based on the cluster count, number of tuples assigned to all the clusters, and the distribution of the result tuples across all the clusters. Using such a comparison, we rank the result variables (or the *ctypes*, or the trees). We then select the first k result variables, where $k = \lceil \frac{|DistVar(q)|+1}{2} \rceil$. For each of the selected result variables (or ctypes), we

additionally obtain the corresponding rtypes from the query. These ctypes and rtypes form the subset of features which will be used by our clustering algorithm.

Nonetheless, we can also employ any other feature selection algorithm which uses the result tuples alone. [Liu and Yu, 2005] provided a detailed classification of different types of feature selection algorithms and we can choose any another suitable algorithm. Let $\mathtt{V} \subseteq DistVar(q)$ be the set of variables obtained from the feature selection algorithm. Let $FS(q, \mathcal{K}, q(\mathcal{K}))$ be the function which selects the result variables used for clustering i.e. $\mathtt{V} = FS(q, \mathcal{K}, q(\mathcal{K}))$. The set of selected ctypes is denoted by $\mathcal{B}$= $\{\mathcal{B}_i \mid \mathcal{B}_i$ is a ctype of $v_i \in \mathtt{V}\}$. The set of selected rtypes is denoted by $\mathtt{R} = \{r \in RTYPE(v_i, q) \mid v_i \in \mathtt{V}\}$. The types $\mathcal{B} \cup \mathtt{R}$ form the set of features which will be used by our clustering algorithm.

The trees (simulating a sub-hierarchy) of concepts or roles rooted at these types will be produced in the next step.

4.3.3 Constructing Trees corresponding to each type

For each type t_i we can construct a (directed) tree T_i rooted at t_i with the concepts (or roles) subsumed by t_i as the nodes. Each tree is defined by the quadruple (N, E, R, ℓ), where N is the set of nodes, E is the set of directed edges between nodes; edge (n_a, n_b) represents an edge from n_a to n_b, $R \in N$ stands for the root node and ℓ is the labelling function assigning each node (having a list of n identifiers) to a list of n concepts and roles i.e.

$\ell(<\text{p}>) = \ell(\text{p}) = C \mid r$.

$\ell(<\text{L}>) =< \ell(<\text{M}>), \ell(<\text{p}>)>$, if <L>=<M,p>

Here, p is a single identifier. L and M are lists of identifiers with ≥ 2 and ≥ 1 elements respectively. The label of the node N_i is the list of identifiers in the node N_i.

We can easily generate the tree T_i rooted at type t_i using the refinement operators. The nodes of the tree $T_i = (N_i, E_i, t_i^c, \ell_i)$ correspond to the concepts subsumed by t_i^c (and t_i^c itself). $(n_a, n_b) \in E_i$ if $\ell_i(n_b) \in minConcSpec(\ell_i(n_a))$. Similarly, the nodes of the tree $T_j = (N_j, E_j, t_j^r, \ell_j)$ correspond to the roles subsumed by t_j^r (and t_j^r itself). $\exists\, e : (n_a, n_b) \in E_j$ if $\ell_j(n_b) \in minRoleSpec(\ell_j(n_a))$.

We also perform few optimization steps while constructing the cluster tree corresponding to a single type to improve the performance of our algorithm. This optimized procedure to generate a tree T_i corresponding to a type t_i is introduced in Section 4.3.5. To combine two trees rooted at 2 different types, an optimized tree-product operator is also introduced in Section 4.3.5. The clusters formed by our algorithm are the non-empty nodes of the combined cluster tree T (using all the types) which is grouped with the result tuples as described in the next sections.

4.3.4 From trees to groups

Having produced a tree $T = (N, E, R, \ell)$ (which corresponds to hierarchical clusters) the next step is to assign the result tuples to one of the clusters (also called as *grouping*). We perform this incrementally. For each result tuple only the projections of the variables in $\{v_i \in \mathtt{V}\} \cup \{v_i, v_j \mid (v_i, v_j) \in Var(t_j^r, q), \forall t_j^r \in \mathtt{R}, \}$ are considered. First, starting from the root we perform membership checks of instances to concepts and roles in the tree nodes to find the (most specific) node under which we place the result tuple.

The population (or grouping) algorithm proposed in [Amato *et al.*, 2010] can be adopted and it has the complexity $\mathrm{O}(nkb^m)$, where n represents the number of answers, k the number of variables in $\mathtt{V}$, and where b is the branching factor and m is the maximum depth of the tree. However, other populating approaches like those using most specific concept can also be applied. To optimize the performance of the grouping algorithm, the leaf nodes which are not assigned with at least one result tuple are deleted from the tree. In addition, we mark the nodes which are assigned with all the result tuples in an intermediate step, $marked(n) \leftarrow true$ and these nodes will not be considered while generating the clusters from the tree.

Let $GROUP(T, q(\mathcal{K}))$ be the function which performs the grouping of the query results tuples $q(\mathcal{K})$ in the tree T. It also removes the leaf nodes in T which generate empty clusters. The list of groups obtained from $GROUP(T, q(\mathcal{K})$ are represented by $G = [G_1, G_2, \dots]$. $GET(G_i, T)$ returns the corresponding node of the group G_i in the tree T.

4.3.5 Generation of the Combined Tree and Optimization

To increase the execution time of our algorithm various optimizations are performed. One such optimization is to integrate (a) generation of the combined tree and (b) grouping of the tree. Moreover, we do not have to perform complete grouping of the extended tree each time and the grouping of the previous un-extended tree can be utilized.

Cluster Tree for a single type

A basic approach to generate the tree T_i of concepts rooted at a ctype $t_i^c \in \mathcal{B}$ is to reproduce (and simulate) a part of the subsumption hierarchy. A similar approach can also be used for generating trees rooted at a rtype $t_i^r \in \mathtt{R}$. In case of complex concepts and roles, the functions $minConcSpec(C)$, $minRolSpec(r)$ defined in Section 3.2 will be used as concept, role refinement operators respectively and are employed while constructing the trees.

While generating a tree corresponding to a type t, all the result tuples can be assigned to a (non-root) node n after grouping the result tuples (as shown in

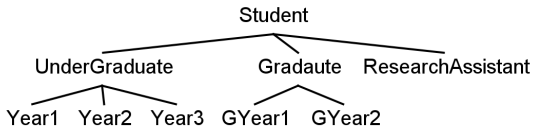

Figure 4.6: Sample tree of a single type: Student

Example 20). In such cases, the type t is replaced with the identifier t' in the node n and we restart the construction of the tree rooted at t' .

Example 20. *Consider the query*
$q(x, y) := Student(x) \wedge takesCourse(x, IntroToUnderGraduateStudies)$
Suppose, using a KB $\mathcal{K}$ the clustering algorithm generated the cluster tree in Figure 4.6. Only 1 ctype, Student, and no rtypes are obtained from q.
After grouping, all the result tuples can fall into the node $Year1$. Then, we set $Year1$ as the root of the tree and continue generating the tree.

Consider that $DEPTH(T)$ gives the depth of the tree T and the function $SUBTREE\_DEPTH(T, d)$ constructs the sub-tree of T with depth d, where $1 < d < DEPTH(T)$. Let, $NODES\text{-}IN(T, i)$ be a function which generates an ordered list of the nodes, $n_1, n_2, \ldots, n_k$, in the tree T which are connected to the root of T through $(i-1)$ nodes i.e. the nodes in T at the depth i, where $1 < i < DEPTH(T)$. The ordering is based on the number of tuples assigned to each node, $TUPLES(T, n_i) > TUPLES(T, n_j)$, $\forall i > j$. $TUPLES(T, n)$ gives the number of result tuples assigned to the node n of T.

Let, **CL** be the approximate number of clusters requested by the user and $|G|$ represents the number of clusters (non-root nodes) $G_1, G_2, \ldots$ generated after grouping. $|G_i|$ represents the number of tuples assigned to the cluster G_i. $REFINE\_OPR(C,\mathcal{K})$ generates the concepts (or roles) that are directly subsumed by C i.e. the set of minimal specializations of C w.r.t. $\mathcal{K}$.

Algorithm: STree
input : *type* $t_i, q(\mathcal{K})$
output : $T_i = (N_i, E_i, R_i, \ell_i)$
1: $E_i \leftarrow \emptyset$
2: $\ell_i(R_i) \leftarrow t_i$
3: $N_i \leftarrow \{R_i\}$
4: $l \leftarrow 1$
5: $cont \leftarrow true$
6: **while** *cont*
7: $\quad l \leftarrow l + 1$
8: $\quad cont \leftarrow false$
9: $\quad$ **for each node** $n \in NODES\text{-}IN(T_i, l-1)$
10: $\quad\quad$ **if** $TUPLES(T_i, n) > M$

```
11:              N_spec ← REFINE_OPR(ℓ_i(n), K)
12:              if N_spec = ∅
13:                  BREAK
14:              for each C ∈ N_spec
15:                  N_i ← N_i ∪ {m}
16:                  ℓ_i(m) ← C
17:                  E_i ← E_i ∪ {(n, m)}
18:                  cont ← true
19:              if cont = true
20:                  G ← GROUP(T, q(K))
21:                  if CL ≤ |G|
22:                      return(N_i, E_i, R_i, ℓ_i)
23:              if |G| = 1 and |G_1| = |q(K)|
24:                  l ← 1
25:                  E_i ← ∅
26:                  ℓ_i(R_i) ← ℓ_i(G_1)
27:                  N_i ← {R_i}
28:                  BREAK
29:              if |G| > 1 and |G_i| = |q(K)|
30:                  marked(Get(G_i)) ← true
31: if(|G| ≥ 1)
32:    return (N_i, E_i, R_i, ℓ_i)
33: else
34:    return NULL
```

Figure 4.7: Generation of a single Tree

So, we use the algorithm **STree** from Figure 4.7 to obtain a single tree corresponding to each type. We can obtain a tree corresponding to one type by iterative application of the concept (or role) refinement operator to the nodes at each level until either (a) we reach the required number of clusters, or (b) we don't extend the tree with a new node. If all the result tuples are assigned to one leaf node t, we rebuild the tree with t as the root as in Steps 23 - 28. Since in $\mathcal{EL}^{++}$ $\exists$ no $\neg$ constructor over concepts and roles, the clusters generated are not mutually exclusive. If the algorithm returns $NULL$, then we ignore the type and continue the tree construction using the next type. If none of the types generate a tree then we obtain 0 c-specializations.

But, to obtain a combined tree for two or more types additional operators like tree-product operator should be introduced. In the next section, we propose the binary operator ''$\otimes$" which takes two trees as arguments and gives a new combined tree as the result.

Cluster tree for multiple types: tree-product operator

Now, let us consider the case of two types, say t_i and t_j used for clustering and their related trees, $T_i = (N_i, E_i, R_i, \ell_i)$ and $T_j = (N_j, E_j, R_j, \ell_j)$. Multiple types will be processed by iterating the binary product operation presented in the Figure 4.8. In cases where the length of the tree T_i is greater than the length of the tree T_j, the leaves in T_j are extended with a dummy node simulating the tree T_i. Thus, we will be able to perform the cross product and traverse atleast one leaf in both the trees as shown in Figure 4.9. **SWAP**(T_i, T_j) assigns T_i to T_j and T_j to T_i. $CHILD(n, T) = \{n' \in N \mid (n, n') \in E\}$ provides the set of nodes having a edge from n in the tree $T = (N, E, R, \ell)$.

$\otimes(T_i, T_j) : (N_{ij}, E_{ij}, R_{ij}, \ell_{ij})$
1: $E_{ij} \leftarrow \emptyset$
2: $R_{ij} \leftarrow < R_i, R_j >$
3: $N_{ij} \leftarrow \{R_{ij}\}$
4: $\ell_{ij}(R_{ij}) \leftarrow < \ell_i(R_i), \ell_j(R_j) >$
5: **if** $DEPTH(T_i) < DEPTH(T_j)$
6: **SWAP** T_i, T_j
7: $d \leftarrow DEPTH(T_i)$
8: **for** $l \in [1 \cdots d]$
9: **for each node** $n_{ij} =< n_i, n_j >, n_i \in NODES\text{-}IN(T_i, l),$
$n_j \in NODES\text{-}IN(T_j, l)$
10: $N_i^{spec} \leftarrow CHILD(n_i, T_i)$
11: $N_j^{spec} \leftarrow CHILD(n_j, T_j)$
12: **if** $N_i^{spec} \cup N_j^{spec} = \emptyset$
13: **continue**
14: **if** $N_i^{spec} = \emptyset$
15: $N_i^{spec} \leftarrow N_i^{spec} \cup n_i$
16: **if** $N_j^{spec} = \emptyset$
17: $N_j^{spec} \leftarrow N_j^{spec} \cup n_j$
18: **for each node** $n_k \in N_i^{spec}$
19: **for each node** $n_l \in N_i^{spec}$
20: $\ell_{ij}(n_{kl}) \leftarrow < \ell_i(n_k), \ell_j(n_l) >$
21: $N_{ij} \leftarrow N_{ij} \cup \{n_{kl}\}$
22: $E_{ij} \leftarrow E_{ij} \cup \{(n_{ij}, n_{kl})\}$
23: **if** $marked(n_k) = true$ && $marked(n_l) = true$
24: $marked(n_{kl}) \leftarrow true$
25: $G \leftarrow Grouping(T', q(\mathcal{K})$
26: **if CL** $\leq |G|$
27: **BREAK**
28:
29: **return** $(N_{ij}, E_{ij}, R_{ij}, \ell_{ij})$

Figure 4.8: Product Operator

The complexity of the operator is polynomial. Moreover, usage of rtypes can be made optional. For this reason, we first combine the trees of ctypes and then combine the trees of rtypes.

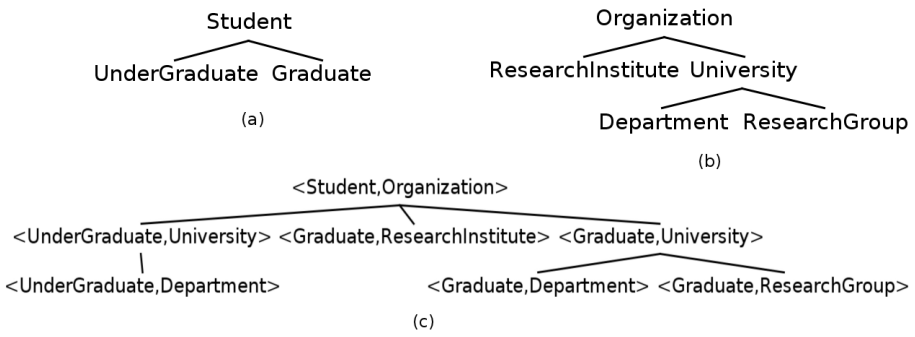

Figure 4.9: A Tree obtained by using the cross product operator $\otimes$. Here, (a) and (b) contain trees with types *Student* and *Organization*. (c) contains the tree obtained by combining trees in (a) and (b) using the product operator using a random KB and some of the nodes are removed in the intermediate grouping steps.

4.3.6 Obtaining c-specializations

The final step in our algorithm is to generate the c-specializations by using the labels of the clusters generated. The clusters' labels are obtained from the nodes of the combined cluster tree. Let q be the original query and $\mathcal{K}$ be the knowledge base. Let, $t_1, t_2, \cdots, t_n$ be the types used for clustering the result tuples $q(\mathcal{K})$ and $L_c = < \cdots < L_{1c}, L_{2c} >, L_{3c} >, \cdots, L_{kc} >, \cdots, L_{(n-1)c} >, L_{nc} >$ be the label of the cluster obtained from a node in the combined tree T, where $0 \leq k \leq n$ and $|\mathtt{V}| = k$. Here, L_{ic} is the i^{th} identifier of the label L_c. In case some of the types are ignored while generating the combined tree then $\mathtt{V}$ is updated accordingly.

$LABEL(T)$ gives the labels of all the non-empty unmarked (i.e. $marked(n) = false$) nodes of the tree T excluding the root node. $TERM(q, i)$ gives the term at the i^{th} position in the query q.

$k\_SPEC(\mathtt{V}, q, T, \mathcal{K}) = \{q_j \mid q_j$ is obtained by replacing the concept or role F in $TERM(q, \mathbf{POS}(t_i, q))$ with $L_{ij}, \forall t_i \in \mathcal{B} \cup \mathtt{R}, \forall L_j \in LABEL(T)\}$

Example 21. *Consider the query*
$q(x, y) := Student(x) \wedge Person(y) \wedge teaches(z, x) \wedge knows(z, y)$
Suppose, using a KB $\mathcal{K}$ the clustering algorithm generated the cluster tree in Figure 4.9 i.e. both x, y are selected for clustering. There are two ctypes Student,

Person corresponding to each of the selected result variable. However, we can't obtain rtypes from q.
The corresponding c-specializations of q w.r.t. $\mathcal{K}$ are
$q_1(x, y) := UnderGraduate(x) \wedge Husband(y) \wedge teaches(z, x) \wedge knows(z, y)$
$q_2(x, y) := UnderGraduate(x) \wedge SingleMale(y) \wedge teaches(z, x) \wedge knows(z, y)$
$q_3(x, y) := UnderGraduate(x) \wedge Wife(y) \wedge teaches(z, x) \wedge knows(z, y)$
$q_4(x, y) := UnderGraduate(x) \wedge SingleFemale(y) \wedge teaches(z, x) \wedge knows(z, y)$
$q_5(x, y) := Graduate(x) \wedge Husband(y) \wedge teaches(z, x) \wedge knows(z, y)$
and so on.

Lemma 4.1. *If $q' \in k\_SPEC(\mathtt{V}, q, T, \mathcal{K})$, then q' is a c-specialization.*

Proof. Assume, a query $q' \in k\_SPEC(\mathtt{V}, q, T, \mathcal{K})$.
We prove, q' is also a c-specialization of q w.r.t. $\mathcal{K}$, by showing that q' satisfies all the three condition specified in Definition 5.2.
Condition 1: Amongst all the nodes in the tree T, none of the nodes containing all the results tuples (at an intermediate step) are considered as clusters. So, none of the queries obtained from the clusters is equivalent to q. i.e. $q'(\mathcal{K}) \subset q(\mathcal{K})$ and condition 1 is satisfied.
Condition 2: We select only the nodes which are non-empty. So, none of the queries obtained from the node labels has empty result set. So, $q'(\mathcal{K}) \neq \emptyset$ and condition 2 is satisfied.
Condition 3: The query q' is obtained by changing the concepts and roles occurring in the terms of q. This implies that q' satisfies condition 3.

Hence, q' is a c-specialization of q w.r.t. $\mathcal{K}$. □

The query results of each of the c-specializations obtained in Example 21 may not be mutually exclusive. Now, we present our complete algorithm which reduces information overload of query results over a DL KB by clustering the query results into approximately n clusters.

4.4 Algorithm: CLUSTERR

The main steps required to generate clusters of query results are explained in the previous sections. CLUSTERR (**C**ontrained divisive hierarchica**L** cl**UST**ering of qu**ER**y **R**esults) first clusters the result tuples of $q(\mathcal{K})$ automatically and produces the corresponding c-specializations. Then, it presents the set of c-specializations to the user for selection and ultimately provides a specialized query result. Figure 4.10 contains the algorithm CLUSTERR. A result tuple is of the form $(a_1, a_2, \ldots, a_k)$ where $a_1, a_2, \ldots, a_n \in Indv(\mathcal{A})$ and $k \geq 1$. We assume that for any variable $x \in DistVar(q)$, there exists only one concept term containing x.

Algorithm: CLUSTERR
input: query q, KB $\mathcal{K}$, $q(\mathcal{K})$

output: specialization of the qw.r.t. $\mathcal{K}$.

$1: \mathtt{V} = FS(q, \mathcal{K}, q(\mathcal{K}))$
$2: \mathtt{S} \leftarrow \emptyset$
$3:$ **for each** $v_i \in \mathtt{V}$
$4:$ $\quad \mathtt{S} = \mathtt{S} \cup \{t_j \mid t_j \in \mathbf{TYPE}(v_i, q)\}$
$5: T \leftarrow \emptyset$
$6:$ *count* $\leftarrow 1$
$7:$ **while** $t_i \in \mathtt{S}$
$8:$ $\quad T^{'} \leftarrow T$
$9:$ $\quad T_i \leftarrow \emptyset$
$10:$ $\quad N_i \leftarrow N_i \cup \{n\}$
$11:$ $\quad \ell_i(n) \leftarrow t_i$
$12:$ $\quad R_i \leftarrow n$
$13:$ $\quad E_i \leftarrow \emptyset$
$14:$ $\quad$ **if** *count* $= 1$
$15:$ $\quad\quad T \leftarrow \mathbf{STree}(T_i, q)$
$16:$ $\quad$ **else**
$17:$ $\quad\quad T \leftarrow \otimes(T, \mathbf{STree}(T_i, q))$
$18:$ $\quad$ *count* $=$ *count* $+ 1$
$19:$ $Q \leftarrow k\_SPEC(\mathtt{V}, q, T, \mathcal{K})$
$15:$ **Obtain the queries** $q_1, \ldots, q_k \in Q$ **selected by the user**
$16:$ **return** $\cup_{i=1}^{k} q_i(\mathcal{K})$.

Figure 4.10: Algorithm CLUSTERR

We can also make the algorithm unsupervised by automatically selecting the clusters (or c-specializations) to be ignored and due to this number of clusters need not be specified too. But the side effect of possible loss of relevant data makes it undesirable. The user is provided with n c-specializations to choose from and once selected, the final specialized query is a union of the selected c-specializations i.e. a union of conjunctive queries.

4.4.1 Complexity

The complexity of our algorithm mainly depends on the complexity of the concept and role refinement operators which is PTIME (as shown in 3.3.2) as we do not perform any CQ containment checks. The complexity of the grouping algorithm is also PTIME and the number of types used for clustering depends on the size of the query $|Q|$. However, the number of the nodes of a tree T is of the order $o(2^{|\mathcal{K}|} * |Q|)$. Hence, the worst-case complexity of the algorithm CLUSTERR is EXPTIME.

Chapter 5

Implementation and Evaluation

In Chapter 3 and Chapter 4 we have presented two approaches to solve the problem of information overload of a query result set over a DL KB. However, there aren't any previous evaluations of a technique to compare our two approaches. In this Chapter, we compare our two approaches (*MinSpec* and CLUSTERR) along with an approach where no processing is done (NO-PROCESSING) for their effectiveness. From the results we show that the two approaches reduce the information overload substantially and we discuss their advantages and limitations if there are any. First, we present the implementation details of the two approaches. Second, we present the benchmark ontology and the test cases upon which the evaluation is conducted. For practical reasons we restrict ourselves to ontologies and queries in regular $\mathcal{EL}^+$. In the end, we present the design and results of the performance evaluation along with the interpretation of the results.

5.1 Implementation

We implemented the two approaches is Java, on top of Pellet[1] reasoner using Jena[2] and OWL API[3]. The experiments were conducted on a laptop computer with a 2.20 GHz Intel Core2 Duo processor, 3GB of RAM, running Ubuntu 11.04. We developed a standalone Java swing application to interact with the user and Figure 5.1 shows the main view of the application.

[1] http://clarkparsia.com/pellet/
[2] http://jena.sourceforge.net/
[3] http://owlapi.sourceforge.net/

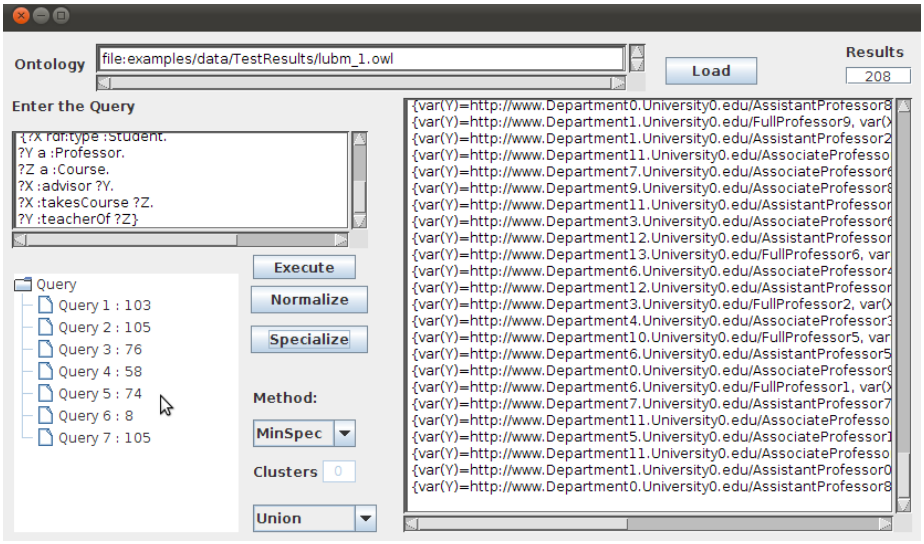

Figure 5.1: Main view of the application to specialize a query

5.1.1 Minimal Syntactic Query Intensification

A specialization of a query is obtained by changing just one syntactic element of the query and the minimal specializations of a query if not avoid information overload can reduce it by a significant amount. In chapter 3, we have presented a computation method *MinSpec* to generate all the minimal specializations of query. Since, this process is automated we restricted ourselves to minimal specializations. Even though the complexity of our algorithm is NPTIME, we could perform various optimizations during implementation for improving the execution time.

Optimization steps

Some of the optimizations used to reduce the execution time are:

- We incrementally store the minimal specializations of a concept C, a role r irrespective of the query i.e. $minConcSpec_{\mathcal{K}}(C)$, $minRoleSpec_{\mathcal{K}}(r)$, *et al.*, where $C \in \mathrm{N}_C^{\mathcal{K}}$, $r \in \mathrm{N}_R^{\mathcal{K}}$. These specializations can be retrieved and employed directly while computing the minimal (concept, role, *et al.*) specializations of the query.

- Among the experiments performed there were no cases, where a minimal specialization of a query is a variable specialization. So, computing these specializations is avoided as it reduces the computation time by a significant amount.

- The search space of concept names and role names for calculating the auxiliary sets like $RC_{\mathcal{K}}$ (in Section 3.2.3) is reduced by ordering the concept and role inclusions in the TBox and RBox respectively.

Here, the query emptiness checks are performed as a final step.

5.1.2 Clustering of Query Results

This approach clusters the query results using the DL KB and the clustering is performed by generating a tree using the features obtained from the query and the KB. The constructed tree resembles a constrained divisive hierarchical clustering structure. The stopping criteria for the constrained divisive hierarchical clustering is based on the number of the clusters **CL** requested by the user. We stop splitting the nodes of the cluster tree if the required number of clusters are grouped. We can further reduce the execution time by using few optimization steps.

Optimization Steps

Major optimization steps of this approach are presented in Section 4.2.6. Further optimization techniques from implementation point of view are

- When we cluster using 2 or more types, the number of nodes generated in each type's tree is decided by using (i) **CL**, (ii) t, the number of types being used for clustering, (iii) i, the number of trees already combined. One such measure to calculate the cluster limit at i^{th} iteration is $\mathbf{CL}_i = \mathbf{CL} * \frac{t+i}{2*t}$.
- A second optimization is instead of grouping the cluster tree after generating it completely, we can just group the intermediate tree accumulated after splitting a node N. Here, only grouping of the sub-tree rooted at the node N using tuples assigned to N is sufficient. Due to this, the number of additional clusters generated and membership checks can be minimized.
- The node selected for splitting is chosen based on the number of tuples assigned to the node. Moreover, we restrict the nodes available for splitting to those which have atleast M tuples assigned to it, where $M = \frac{|q(\mathcal{K})|}{2*CL}$. This ensures that the final set of clusters generated are of approximately the same size.

5.2 Evaluation Test Data

We empirically test our two approaches using the benchmark dataset LUBM$_{\mathcal{EL}^+}$ (an extension of Lehigh University Benchmark, LUBM[4]). We select LUBM because, first of all, it is quite popular and widely used conjunctive query answering benchmark. Secondly, most of the other $\mathcal{EL}^+$ benchmark ontologies do not

[4]http://swat.cse.lehigh.edu/projects/lubm/

Table 5.1: The list of queries used in the evaluation

Names	Queries	Results
Query 1	$q_1(x,y) \leftarrow Faculty(x) \wedge worksFor(x,y) \wedge Organization(y)$	541
Query 2	$q_2(x,y,z) \leftarrow Student(x) \wedge Faculty(y) \wedge Course(z) \wedge advisor(x,y) \wedge takesCourse(x,z) \wedge teacherOf(y,z)$	208
Query 3	$q_3(x,y) \leftarrow Person(x) \wedge worksFor(x,y)$	16666
Query 4	$q_4(x,y) \leftarrow Person(x) \wedge publicationAuthor(y,x) \wedge Work(y)$	30
Query 5	$q_5(x) \leftarrow Person \sqcap takesCourse.GraduateCourse(x) \wedge worksFor(x,y) \wedge Organization(y)$	547

contain ABox assertions, which makes them unsuitable for our purpose. Alternatively, LUBM already has an automatic ABox generation mechanism. Last by not the least, the language of LUBM ($\mathcal{ELHID}$) is quite close to $\mathcal{EL}^+$ and we slightly modify LUBM to meet the $\mathcal{EL}^+$ expressive power as shown in [Zhao *et al.*, 2009] to obtain LUBM$_{\mathcal{EL}^+}$. The ontology we used has 17210 individuals, 25 roles, 43 concepts, 38 concept inclusions, 5 simple role inclusions and 1 complex role inclusion.

Both the approaches are tested using the 5 queries in Table 5.1. For each query, we choose 5 subsets of results tuples as relevant (or irrelevant) for the user to test the 3 approaches (*MinSpec*, CLUSTERR, NO-PROCESSING). These subsets can be perceived as the answers to a query obtained by changing the concepts and roles in the query. For Query 1, a tuple subset T1:[*Lecturer*,*Department*] corresponds to all the result tuples of the query

$$q(x,y) \leftarrow Lecturer(x) \wedge worksFor(x,y) \wedge Department(y).$$

i.e. the results of the query obtained by appropriately replacing with *Lecturer* and *Department* in Query 1. This is only one way to obtain the subsets tuples assuming that the user's selection of result tuples is based on the concept and role membership likelihood of the individuals occurring in the tuples. Table 5.2 provides a complete list of all the subset of result tuples selected as relevant (or irrelevant) for the user in our test cases for each query.

5.3 Evaluation on Quality and Performance

The main goal of this evaluation is to present the use of our two approaches for reducing information overload of query results over a DL knowledge base. As a pre-processing step, both the approaches first perform a rewriting of the given query by converting it into CQNF. The post-processing steps eliminate the new variables introduced (in the rewriting steps) resulting in a concise query. However, in *MinSpec* new role terms are introduced to generate a specialization which can lead to the increase in the size (number of terms) of a specialization.

Table 5.2: Result tuple subsets used for each query

Query	T1	T2	T3	T4	T5
Query 1	[AP,Dep]	[$Lect$,Dep]	[$Prof$]	[FP]	[$Dean, Dep$]
Query 2	[UgS]	[AP]	[GC]	[$Chair$,GS]	[FP,UgS]
Query 3	[Std]	[$GS, mem \circ sOrg$]	[UgS]	[TA]	[$Emp, wrkFr \circ sOrg$]
Query 4	[FP]	[$Prof$,$Course$]	[$Lect$]	[AiP,$Rsch$]	[FP,$Course$]

Here, AP is $AssociateProfessor$, AiP is $AssistantProfessor$, Dep is $Department$, Emp is $Employee$, FP is $FullProfessor$, GC is $GraduateCourse$, GS is $GraduateStudent$, $Lect$ is $Lecturer$, mem is $memberOf$, $Prof$ is $Professor$, $Rsch$ is $Research$, Std is $Student$, $sOrg$ is $subOrganizationOf$, TA is $TeachingAssistant$, UgS is $UndergraduateStudent$, $wrkFr$ is $worksFor$, $wrkFr \circ sOrg$ represents $worksFor \circ subOranizationOf$, $mem \circ sOrg$ represents $memberOf \circ subOrganizationOf$.

The question, "How well does the system works?", can be investigated at several levels

(a) Search: Effectiveness of the results
(b) Processing based: Time and space efficiency
(c) System: Satisfaction of the user

Here, we measure the effectiveness of the queries generated by each approach along with their computing time for each query. By performing various optimizations and pre-processing steps like, the calculation of the minimal specializations of all concept names and role names in the KB, generation of role predecessor and role successor hierarchy forest, *et al.* we can reduce the computation time (of both the methods) substantially. Investigating the satisfaction of the user requires a real knowledge base (including an ABox) which is not available. However, the sample subset tuples (in Table 5.2) selected for each query can be viewed as a simulated input for user's relevant (or irrelevant) tuples.

5.3.1 Effectiveness measure

The effectiveness of a result depends on how well the problem of information overload is solved. The amount of reduction in the information overload is evaluated using an effectiveness measure. Let, $\mathtt{T} \subset q(\mathcal{K})$, be a set of result tuples. We calculate the effectiveness measure **E** using

1. *Precision* **P** of the optimal query obtained from the (c-)specializations, when the set of result tuples T are relevant for the user.
2. *Recall* **R** of the optimal query obtained from the (c-)specializations, when the set of result tuples T are irrelevant for the user.
3. Number of (c-)specializations, k_1, k_2, required to obtain the values **P**, **R** respectively.

Here, *Precision* is the probability that the retrieved tuples are relevant and *Recall* is the probability that the relevant tuples are retrieved in a search. Since

each approach can provide more than one (c-)specializations and we can combine these specializations in various ways to obtain different queries, we represent the appropriate (and most specific) query containing the result tuple subset T as the optimal query. Let, n be the number of (c-)specializations available to the user for selection. The formula for the effectiveness measure is given by:

$$\mathbf{E = (P + R)/2} * \sqrt{\frac{2*n+1-(k_1+k_2)}{2*n-1}},$$

The value of **E** increases as the precision and recall of a query obtained (c-)specializations w.r.t. the tuple subset T increases. However, by computing huge number of clusters (and selecting just some of the clusters) we could also achieve higher precision and recall. So, to avoid such situations, we also take the number of (c-)specializations generated, n, and the number of (c-)specializations utilized k_1, k_2 into considered. When the values of **P, R**, k_1, k_2 are all 1, **E** is reaches the value of 1. As the value of **E** increases the information overload decrease and when $\mathbf{E} = 1$, the information overload is completely avoided.

The set of tuples which are relevant or irrelevant varies between different users because of which we use 5 different tuple sets for our evaluation. The evaluation is performed using the algorithm in Figure 5.2.

Step 1: For each query $q \in \{q_1, q_2, q_3, q_4, q_5\}$
Step 2: Generate specializations of q: $spec = MinSpec_{\mathcal{K}}(q)$
Step 3: Generate c-specializations of q: $c\text{-}spec$ = CLUSTERR$(q, q(\mathcal{K}))$
Step 4: For each tuple subset $\mathtt{T} \in \{\mathtt{T}_1, \mathtt{T}_2, \mathtt{T}_3, \mathtt{T}_4, \mathtt{T}_5\}$ w.r.t. q
Step 5: Calculate effectiveness measure E_s of *spec*
Step 6: Calculate effectiveness measure E_c of *c-spec*
Step 7: Calculate effectiveness measure E_o of q

Figure 5.2: Effectiveness measure algorithm

In addition, we can iterate the computation of $MinSpec$, CLUSTERR to further reduce the information overload. As, the number of such iterations increase, the effectiveness of the specializations also increase and after few iterations **E** reaches 1 when $\mathbf{P} = 1, \mathbf{R} = 1, k_1 = 1, k_2 = 1$. *So, we also compare the number of iterations needed for each approach to obtain an effectiveness measure of 1 for each tuple subset* T.

5.3.2 Results and Interpretation

Now, we provide the results corresponding to each of the first 4 queries considered. The 3 approaches $MinSpec$, CLUSTERR, NO-PROCESSING are compared for the queries Query 1, Query 2, Query 3, Query 4 with 5 subset of relevant (or irrelevant) result tuples.

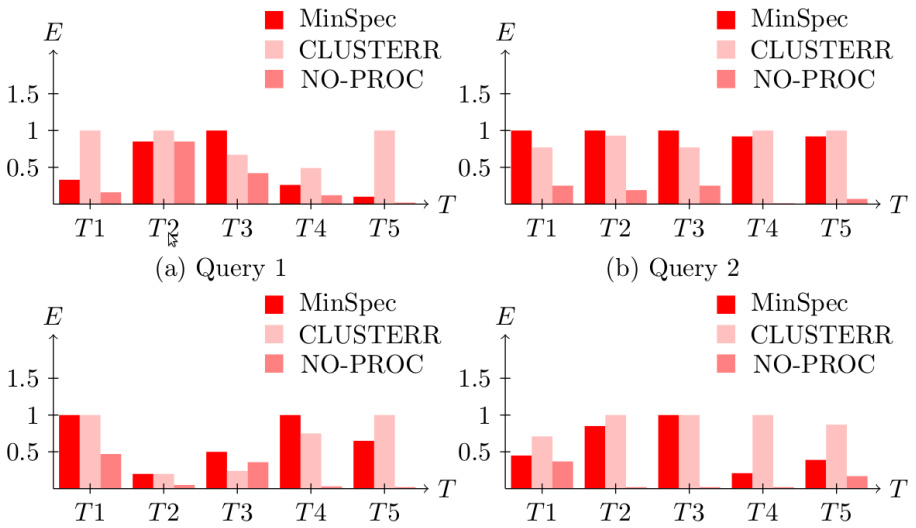

Figure 5.3: Effectiveness measure of the 4 queries using the 3 approaches

Interpretation of the results of effectiveness measure experiments

In Figure 5.3, the calculated effectiveness measure of Query 1, Query 2, Query 3 and Query 4 is provided (NO-PROC represents NO-PROCESSING approach). The subset of (relevant/irrelevant) result tuples $T1, T2, T3, T4, T5$ of each query are selected randomly (simulating a random choice of users) using the KB. Due to the limitations of the KB, *MinSpec* and CLUSTERR may not compute any specialization and c-specialization respectively for some queries like Query 5. In cases like these techniques developed for Databases can be adopted.

From Figure 5.3, we can see that *MinSpec* and CLUSTERR improve the effectiveness of the search (reduce information overload) in virtually every test case. In some cases, the improvement can be huge as shown in the results acquired for Query 4. Between *MinSpec* and CLUSTERR, CLUSTERR provides better results in most of the test cases. This is not unexpected as CLUSTERR's c-specializations can reproduce most of the specializations generated by *MinSpec* and it is evident from the results of Query 1, Query 2 and Query 4 where 11 out of 15 times CLUSTERR generates more effective results.

However, in cases where the number of results variables is very small or if some of the tuples are contained in none of the clusters generated, *MinSpec* performs better. For Query 3, CLUSTERR provides better results than *MinSpec* in one case only. Moreover, *MinSpec* can compute various kinds of specializations like variable individual specialization, individual specializations *et al.*, which is not possible with CLUSTERR. These specializations can be significant especially when the KB doesn't provide sufficient information (concept and role inclu-

sions) to perform further specialization. However, all these cases are still only isolated and on an average CLUSTERR provides better results than $MinSpec$.

One interesting revelation from the experiments is the ability of role compositions to generate a decent cluster tree from a role (rtype) even with few role inclusions. For Query 3, a cluster tree with $memberOf$ as root is extended to several levels. Since the role inclusion $memberOf \circ subOrgnizationOf \sqsubseteq memberOf$ is added to LUBM to obtain $\text{LUBM}_{\mathcal{EL}^+}$ and $worksFor \sqsubseteq memberOf$ already exists in LUBM, we can infer the following inclusions

$memberOf \circ subOrganizationOf \circ subOrganizationOf \sqsubseteq memberOf \circ subOrgnizationOf$,
$worksFor \circ subOrganizationOf \sqsubseteq memberOf \circ subOrgnizationOf \sqsubseteq memberOf$

and a decent cluster tree can be obtained. However, once groping is performed empty nodes are eliminated and this restricts the expansion of the cluster tree.

Additional specifics ascertained

The various steps in the clustering algorithm like feature selection are also highly important. In Query 2, only $Student$ and $Faculty$ are used as features and we ignored $Course$ as a feature. Since only one concept is subsumed by $Course$ ($GraduateCourse$) in our test ontology, using $Course$ as a feature reduces the number of tuples assigned to the non-root nodes of the cluster tree. This leads to the formation of non-optimal clusters which can be avoided by careful selection of features. Even though we populate hierarchical clusters, it is not always plausible to compute the exact number of clusters as specified by the user. Otherwise in some cases, this can lead to omission of some of the result tuples having an adverse effect on the effectiveness and quality of the result. So, our method only forms approximately n number of clusters, where n is the number of clusters suggested by the user.

For $MinSpec$, obtaining individual specializations, variable individual specializations, *et al.*, which are minimal is certainly possible even for a small KB that we experimented with. For the query
$q(x, y) \leftarrow Publication(x) \wedge publicationAuthor(x, y) \wedge Employee(y)$
we obtained the following query as a minimal specialization
$q_1(x, y) \leftarrow Publication(x) \wedge publicationAuthor(Publication9, y) \wedge Employee(y)$,
where $Puglication9$ is an individual occurring in the KB.

Iterations required to avoid information overload

Even though the effectiveness of the (c-)specializations obtained from $MinSpec$, CLUSTERR is improved substantially, the information overload may not be completely avoided. To obtain the effectiveness measure of 1 (or to avoid information overload), we can perform more iterations by specializing the appropriate (c-)specialization(s). The change in the effectiveness for each query after each

iteration is shown in the Figure 5.4 (Itr i represents Iteration i). As displayed in the test results, CLUSTERR once again performs better than *MinSpec* and achieves optimal effectiveness with fewer iterations. CLUSTERR requires at most 2 iterations to obtain effectiveness measure $E = 1$ in all the test cases, while *MinSpec* requires 3 iterations in some of the test cases.

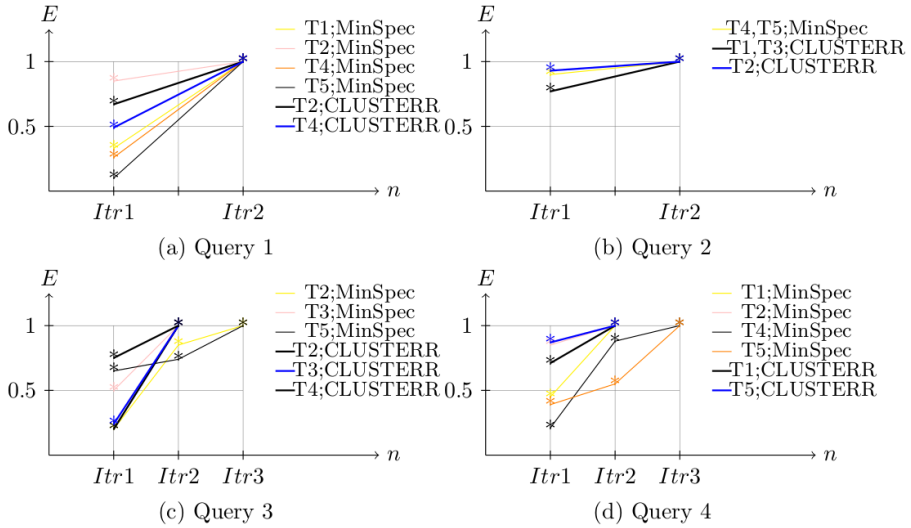

Figure 5.4: Increase in effectiveness after each iteration for the two approaches

In Figure 5.4 only the test cases which didn't have value 1 for the effectiveness measure in the first iteration are considered.

5.3.3 Evaluation of Computation Time

To evaluate the time efficiency of our two approaches we measured the amount of time taken by *MinSpec* and CLUSTERR to compute the specializations and c-specializations respectively of each of the above 5 queries. We also included the time required to normalize the query into CQNF. From the Table 5.3, we can see that *MinSpec* takes more time than CLUSTERR as it computes the complete set of minimal specializations of a query unlike CLUSTERR and it also performs query equality and emptiness checks to satisfy the conditions 1 and 2 in Definition 3.1. However, if we assume that all the variables in the query are distinguished variables, then the speed of the algorithm *MinSpec* increases substantially as we only perform membership checks.

The values corresponding to *MinSpec*, CLUSTERR in Table 5.3 are achieved when the minimal specializations of a concept (irrespective of the query) are computed either as a pre-processing step or incrementally so that they are readily accessible when requested by our methods. Due to this, the speed of the algorithm CLUSTERR mainly depends on the number of result tuples of the

Table 5.3: Computation time for *MinSpec* and CLUSTERR

Query	Results	Normalize(ms)	*MinSpec*(ms)	CLUSTERR(ms)
Query 1	541	1	240	76
Query 2	208	1	192	39
Query 3	16666	1	384	2153
Query 4	30	1	126	35
Query 5	547	2	470	50

original query and not on the size of the knowledge base. In addition, we allowed empty queries to be included in the results obtained from *MinSpec*. In case of queries like Query 3 where the size of the result set is 16666, the computation times of *MinSpec* and CLUSTERR are still around 2 seconds. This shows that, both *MinSpec* and CLUSTERR are scalable even for large KBs, once the minimal specializations of all the concept and role names occurring in the KB are pre-computed.

Conclusion

Information overload is one of the main problems which should be effectively tackled by every search and query engine, be it Databases or DL KBs or the World Wide Web (WWW). Various studies aimed at avoiding (or reducing) information overload in web-based search and Database querying resulted in the development of numerous techniques based on ranking, clustering, classification. However, employing those techniques directly to DL KBs wouldn't yield optimal results. In addition, the increase in the usage of ontologies and Semantic Web also aggravated the need for new customized techniques for DL KBs to reduce information overload. In this thesis, we presented two approaches tailored for DL (especially $\mathcal{EL}$ family) KBs to address our problem, reducing information overload. For better understandability, we assumed that all KBs are in a normal form [Baader *et al.*, 2005].

In Chapter 3, we presented our first approach which is based on minimal query intensification such that the newly generated query is syntactically similar to the original query. These syntactically similar queries obtained after intensification are called *specializations*. We have also shown that *minimal specializations* are sufficient to attain decent reduction in information overload. We presented various ways to obtain specializations and provided an algorithm to compute the set of all minimal specializations of a query. The soundness and completeness of our method $MinSpec$ is also proved along with termination. It is also shown that the combined complexity of $MinSpec$ is NPTIME as the most complex steps, subsumption checking and conjunctive query containment, are performed polynomial number of times only.

In Chapter 4, we presented our second approach CLUSTERR which is based on clustering of query results. Unlike $MinSpec$, CLUSTERR also acquires queries that are non-minimal. Using the query and the KB, we constructed a cluster tree which generates hierarchical clusters upon populating it with the result tuples. From these clusters and their labels we can generate new queries, called *c-specializationss*, having these clusters as their result sets. Various shortcomings of other clustering based approaches like [Amato *et al.*, 2010] e.g. unavailability of a concept and role refinement operator, non-utilization of the role inclusions available in the KB are recognised and dealt in our approach. The main steps of the algorithm like grouping of the tree, tree product opera-

tion are performed within PTIME, but the generation of a single tree requires EXPTIME. So,the complexity of the algorithm CLUSTERR is EXPTIME.

We also implemented the above two algorithms *MinSpec*, CLUSTERR in Java on top of Pellet reasoner using Jena and OWL APIs. We evaluated the two approaches based on the effectiveness of the (c-)specializations computed by each method using an extended LUBM benchmark ontology. The experiments clearly show that both methods reduce the information overload by a significant amount. Between *MinSpec* and CLUSTERR, on an average the later performs better. The processing time of each method is improved immensely by pre-computing the minimal specializations of the concept and role names occurring in the KB. Even though the complexity of our methods *MinSpec* and CLUSTERR is NPTIME and EXPTIME respectively, after performing the pre-processing steps and optimizations *MinSpec* and CLUSTERR are scalable even for large KBs. The EXPTIME complexity of CLUSTERR is acceptable, because in most of the KBs the ratio of concept (or role) inclusions to concepts (or roles) is very low.

Open Issues

The method *MinSpec* is highly efficient when all the variables occurring in the query are distinguished variables as instead of query equivalence and emptiness checks, it is sufficient to perform membership checks only. However, in the presence of non-distinguished variables, checking if a concept is a minimal specialization w.r.t. the query becomes a harder problem as query equivalence and emptiness has to be checked to avoid inclusion of non-specializations. Additionally, when variable specializations are also employed we might also perform query containment checks and optimized solutions can be developed as a future work.

The number of terms in the specializations of a query can be higher when compared to the original query. This is understandable as conjunct specializations are also utilized. In this thesis, we assumed that concrete domains do not occur in our KBs and their inclusion makes our problem slightly harder. We also limited ourselves to regular-$\mathcal{EL}^{++}$ KBs, and as a future work, *MinSpec* and CLUSTERR can be extended and implemented for KBs which are expressed using unrestricted $\mathcal{EL}^{++}$. Extending CLUSTERR to other DLs is fairly simple once we compute the concept and role refinement operators. However, extending *MinSpec* to other DLs requires much more work but it is still possible.

The clusters (and c-specializations) generated by CLUSTERR form a hierarchical clustering structure with the root of the cluster tree containing all the result tuples. However, in few cases some of the result tuples might be assigned to the root node only and these result tuples belong to the result set of none of the c-specializations computed. Moreover, in some cases our two approaches

may not generate any (c-)specializations and using the techniques developed for Database technologies is more appropriate.

Automating the calculation of the optimal number of clusters is a first-step towards an unsupervised method for clustering query results. As a next step, we can select the clusters automatically to generate the specialized query (and result set). A similar step can also be performed in $MinSpec$ by automatically selecting one or more minimal specializations to generate the specialized query. The techniques developed for Databases and WWW can be directly utilized here and further extensions to our approaches can be done. In the method CLUSTERR, we only introduced the idea of selecting the result variables for clustering w.r.t. a DL KB using a naive technique (Section 4.3.1). More advanced techniques to perform this selection, $FS(q, \mathcal{K}, q(\mathcal{K}))$, needs to be developed.

Bibliography

[Abiteboul *et al.*, 1995] S. Abiteboul, R. B. Hull and V. Vianu, "Foundations of Databases", *Addison-Wesley*, 1995.

[Agarwal *et al.*, 2003] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, and Aristides Gionis, "Automated ranking of database query results", in $CIDR$, 2003.

[Amato *et al.*, 2010] C. d'Amato, N. Fanizzi, and A. Lawrynowicz, "Categorize by: Deductive Aggregation of Semantic Web Query Results", in *Proc. ESWC (1)*, 2010, pp. 91-105.

[Baader and Nutt, 2003] F. Baader and W. Nutt. The Description Logic Handbook, chapter 2: Basic Description Logics. *Cambridge University Press*, 2003.

[Baader *et al.*, 2005] F. Baader, S. Brandt and C. Lutz, "Pushing the $\mathcal{EL}$ envelope". In *Proc. of IJCAI 2005*, pp. 364-369.

[Bosc *et al.*, 2008] P. Bosc, A. Hadjali, and O. Pivert, "Empty versus over-abundant answers to flexible relational queries", presented at *Fuzzy Sets and Systems*, 2008, pp. 1450-1467.

[Cai *et al.*, 2010] D. Cai, C. Zhang, X. He, "Unsupervised feature selection for multi-cluster data", in *Proc. of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, July 25-28, 2010, Washington, DC, USA.

[Calado *et al.*, 2003] P. Calado, M. Cristo, E. S. D. Moura, N. Ziviani, B. A. Ribeiro-Neto, and M. A. Goncalves, "Combining link-based and contentbased methods for web document classification". In *Proc. of CIKM-03, 12th ACM International Conference on Information and Knowledge Management, New Orleans, US,* 2003, pp. 394-401.

[Chakrabarti *et al.*, 2004] K. Chakrabarti, S. Chaudhuri, and S. W. Hwang, "Automatic categorization of query results", in *SIGMOD Conference*, 2004, pp. 755-766.

[Chandra and Merlin, 1977] A. K. Chandra and P. M. Merlin, "Optimal implementation of conjunctive queries in relational data bases". In *Proc. of the 9th ACM Symp. on Theory of Computing* ($STOC'77$), pp. 77-90.

[Chu *et al.*, 1996] W.W. Chu, K. Chiang, C. Hsu, and H. Yau, "An Error-based Conceptual Clustering Method for Providing Approximate Query Answers", presented at *Commun. ACM*, 1996, pp. 216-230.

[Corent and Keizer, 2008] R. Cornet and N. de Keizer, "Forty years of SNOMED: a literature review". BMC medical informatics and decision making, 2008.

[Daniels, 2006] K. J. Daniels, "Clustering of database query results", Master Thesis, Department of Computer Science, Brigham Young University, 2006.

[Dash and Liu, 1997] M. Dash and H. Liu, "Feature Selection for Classification", in *Intelligent Data Analysis* vol. 1, no. 3, pp. 131-156, 1997.

[Fu *et al.*, 2004] L. Fu, D. Goh, S. Foo, "Query clustering using a hybrid query similarity measure". In *WSEAS Transactions on Computers, 3(3)*, 2004, pp. 700-705.

[Gene Ontology, 2000] The Gene Ontology Consortium, "Gene Ontology: Tool for the Unification of Biology". In *Journal of Nature Genetics, 25* , pp. 25-29, 2000.

[Glimm *et al.*, 2007] B. Glimm, I. Horrocks, C. Lutz and U. Sattler, "Conjunctive Query Answering for the Description Logic $\mathcal{SHIQ}$". In *Proc. IJCAI*, 2007, pp. 399-404.

[Glimm and Rudolph, 2010] B. Glimm, S. Rudolph, "Status $\mathcal{QIO}$: Conjunctive Query Entailment is Decidable", in $KR'2010$.

[Grira *et al.*, 2004] N. Grira, M. Crucianu and N. Boujemaa, "Unsupervised and Semi-supervised Clustering: a Brief Survey". In *A Review of Machine Learning Techniques for Processing Multimedia Content, Report of the MUSCLE European Network of Excellence (FP6)*, 2004.

[Hammouda and Kamel, 2004] K. M. Hammouda and M. S. Kamel, "Efficient phrase-based document indexing for web document clustering". In *IEEE Trans. Knowl. Data Eng*, vol. 16, no. 10, pp. 1279-1296, 2004.

[Kang and Kim, 2003] In-Ho Kang and Gil-Chang Kim, "Query type classification for web document retrieval", in $SIGIR$, 2003, pp. 64-71.

[Kolaitis and Vardi, 2000] P. G. Kolaitis and M. Y. Vardi, "Conjunctive-Query Containment and Constraint Satisfaction", in *Journal of Computer and System Sciences 61*: 302-332, (2000).

[Kroetzsch and Rudolph, 2007] M. Kroetzsch and S. Rudolph, "Conjunctive queries for $\mathcal{EL}$ with role composition". In *Proc. of DL07*, pages 355-362.

[Lawrynowicz, 2009] A. Lawrynowicz, "Grouping results of queries to ontological knowledge bases by conceptual clustering". Nguyen, N.T., Kowalczyk, R., Chen, S.M., eds.: In *ICCCI*. Volume 5796 of Lecture Notes in Computer Science., Springer (2009) 504-515.

[Lawrynowicz, 2009a] A. Lawrynowicz, "Query results clustering by extending sparql with cluster-by". In Meersman, R., Herrero, P., Dillon, T.S., eds.: OTM Workshops. Volume 5872 of Lecture Notes in Computer Science., Springer (2009) pp. 826-835.

[Liu and Yu, 2005] H. Liu and L. Yu, "Toward Integrating Feature Selection Algorithms for Classification and Clustering". In *IEEE Transactions on Knowledge and Data Engineering, 17(4)*, 491-502, 2005.

[Manning *et al.*, 2008] C. D. Manning, P. Raghavan and Hinrich Schuetze, "Introduction to Information Retrieval", Cambridge University Press. 2008 , urlhttp://nlp.stanford.edu/IR-book/html/htmledition/hierarchical-agglomerative-clustering-1.html.

[Prud'hommeaux and Seaborne, 2008] E. Prud'hommeaux, A. Seaborne, "SPARQL Query Language for RDF", Technical report, *W3C Recommendation* (January 15, 2008)

[Rector and Horrocks, 1997] A. Rector and I. Horrocks, "Experience building a large, reusable Medical Ontology using a Description Logic with Transitivity and Concept Inclusions". In *Proc. of the Workshop on Ontological Engineering, AAAI Spring Symposium*, 1997.

[Rosati, 2007] R. Rosati, "On conjunctive query answering in $\mathcal{EL}$". In *Proc. of DL07*, volume 250 of *CEUR-WS*, 2007.

[Savaresi *et al.*, 2002] S. M. Savaresi, D.Boley, S.Bittanti, and G. Gazzaniga, "Cluster Selection in Divisive Clustering Algorithms", in *Proc. SDM*, 2002.

[TheFreeDictionary, 2011] The Free Dictionary, thefreedictionary.com, `http://www.thefreedictionary.com/Clustering`, 2011.

[Zhao *et al.*, 2009] Y. Zhao, J. Z. Pan, Y. Ren, "Implementing and Evaluating a Rule-Based Approach to Querying Regular $\mathcal{EL}^+$ Ontologies". In *Proc. of the International Conference on Hybrid Intelligent Systems, HIS* 2009, pp. 493-498.

[Zhao *et al.*, 2010] Z. Zhao, F. Morstatter, S. Sharma, S. Alelyani, A. Anand, and H. Liu, "Advancing Feature Selection Research - ASU Feature Selection Repository", School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, 2010.