NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
Division: Communication, Electronic and Information Engineering

Query Optimization under bag and bag-set semantics for multiple heterogeneous data sources

Ph.D. Thesis

Matthew Damigos

Athens, March 2011

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
Division: Communication, Electronic and Information Engineering

ADVISORY COMMITTEE

- Principal Advisor:

Foto Afrati
Professor
School of Electrical and Computer Engineering
(National Technical University of Athens)

- Members:

Manolis Gergatsoulis
Associate Professor
Department of Archives and Library Science
(Ionian University)

Yannis Vassiliou
Professor
School of Electrical and Computer Engineering
(National Technical University of Athens)

EXAMINATION COMMITTEE

Timos Sellis
Professor
School of Electrical and Computer Engineering
(National Technical University of Athens)

Panos Rondogiannis
Associate Professor
Department of Informatics and Telecommunications
(University of Athens)

Vassilis Zissimopoulos
Professor
Department of Informatics and Telecommunications
(University of Athens)

Nikolaos Papaspyrou
Assistant Professor
School of Electrical and Computer Engineering
(National Technical University of Athens)

....
Matthew Damigos
Doctor of Electrical and Computer Engineering of N.T.U.A.

Contents

1 Introduction **5**

1.1 Motivation scenarios 8

1.1.1 Organization of thesis 12

2 Querying relational databases using views **15**

2.1 Basic definitions 15

2.1.1 Operators 16

2.1.2 Expressions and substitutions 18

2.2 Queries and Views 23

2.2.1 Semantics of Queries 25

2.3 Containment and Equivalence 27

2.3.1 Deciding conjunctive query containment 30

2.3.2 Conjunctive query equivalence 35

2.3.3 Further related work on query containment and equivalence problems 37

2.4 Answering queries using views 38

2.4.1 Useful views for rewriting a CQ 41

2.4.2 On constructing equivalent rewritings 43

2.5 Selecting conjunctive views 49

2.5.1 Determining the search space of solutions 52

2.5.2 Related work on selecting views 55

3 Query containment under bag and bag-set semantics **57**

3.1 Contained query without projections 58

3.2 Contained Queries without self-joins 61

3.3 Generalized-Pathstar Queries 63

3.3.1 Pathstar queries 63

3.3.2 Simple generalized-pathstar queries 65

3.4 Other syntactic restrictions 68

3.5 Conclusions and Future Work 72

4 View Selection under Bag and Bag-Set semantics 73

4.1 Space of optimal solutions under bag semantics 75

4.1.1 On restricting the space of admissible viewsets 75

4.1.2 Minimum set of distinguished variables in candidate rewritings 81

4.1.3 Representative set of solutions under bag semantics . 84

4.1.4 LGG-VSB Algorithm 86

4.2 Space of Optimal solutions under bag-set semantics 87

4.2.1 Useful viewsets for rewriting CQs under bag-set semantics 89

4.2.2 Representative set of solutions for the bag-set-oriented view selection problem 91

4.3 Chain and Path queries 98

4.3.1 Chain-query workload 99

4.3.2 Path-query workload 102

4.4 Conclusions and Future Work 106

5 Heterogeneous Data Sources (XML data) 109

5.1 XML trees and Patterns 113

5.1.1 Semantics of Patterns 116

5.1.2 Core Pattern 118

5.2 Containment and Equivalence of sets of patterns 120

5.2.1 Deciding containment and equivalence of patterns . . 121

5.2.2 Containment of unions 128

5.2.3 Complexity of containment problem 131

5.2.4 Further related work on containment and equivalence of patterns 132

5.2.5 Answering a pattern using a single view 133

6 Answering patterns using union rewritings 139

6.1 Containment and equivalence of unions of patterns 140

6.1.1 Deciding unions containment in $XP^{\{[\,],*\}}$ 142

6.1.2 Deciding equivalence of patterns in $XP^{\{//,*\}}$ 142

6.1.3 Containment of unions in the fragment of $XP^{\{//,*\}}$ . . 144

6.1.4 Equivalence of a pattern and a union of patterns in $XP^{\{//,*\}}$ 153

6.2 Rewriting pattern using views 155

6.2.1 Solving the problem of the existence of a union rewriting 158

6.3 Conclusions and Future Work 164

7 Conclusions **165**

Bibliography **167**

List of Tables

3.1 Complexity results for the CQ containment problem. 58

5.1 Complexity of containment of XPath fragments 132

Abstract

Nowadays, as the amount of information is rapidly increasing, query optimization techniques for both homogeneous and heterogeneous data are more and more needed. In this thesis, we investigate techniques for query optimization using a set of views, considering both relational and XML databases. In particular, we focus on three fundamental problems of query optimization, which have been extensively investigated due to their relevance with many database research areas. These problems are the query containment, the query rewriting and the view selection.

For relational databases we focus on the class of select-project-join SQL queries with equality comparisons, a.k.a. conjunctive queries (CQs for short); one of the most interesting classes of SQL queries and the class that has been greatly investigated. Moreover, we consider two kinds of semantics in order to theoretically approximate the SQL semantics: the bag and bag-set semantics. In bag semantics multiple occurrences of the same tuple are allowed in both base relations and answers of queries. In bag-set semantics, the base relations are sets and the operators are liable for bag-results.

For querying XML databases, we focus on XPath, which constitutes the major language for navigation through the XML data. In particular, we focus on the three major fragment of $XP^{\{//,[\,],*\}}$, which contain two of the constructs: wildcard ($*$), descendant edge ($//$) and branches ($[\,]$).

The problems of query containment and equivalence under both bag and bag-set semantics are investigated through a detailed analysis of special cases of CQs. The complexity in each case is given, as well. For the general case, the problem remains open for more than a decade. In addition, we give necessary and sufficient conditions for deciding both containment and equivalence for unions of XPath queries; a problem which is not investigated in depth, in the past.

The problem of finding an equivalent rewriting is also investigated for both relational and XPath queries. In particular, for relational queries, we describe the requirements that a set of views have to satisfy in order to

give an equivalent rewriting of a CQ under both bag and bag-set semantics. In the case of XML databases, we investigate the problem of rewriting a XPath query using multiple views and prove that in the case that the query contains both descendant edges and wildcards, the union operator may be required for finding an equivalent rewriting.

The view selection is investigated for workloads of CQs under both bag and bag-set semantics. In particular, we aim to limit the search space of candidate viewsets. In that respect we start delineating the boundary between query workloads for which certain restricted search spaces suffice. They suffice in the sense that they do not compromise optimality in that they contain at least one of the optimal solutions. We start with the general case, where we give a tight condition that candidate views can satisfy and still the search space (thus limited) does contain at least one optimal solution. Then we study special cases. We show that for chain query workloads under both bag and bag-set semantics, taking only chain views may miss all optimum solutions, whereas, if we further limit the queries to be path queries (i.e., chain queries over a single binary relation), then under bag semantics, path views suffice.

Chapter 1

Introduction

The efficient querying of data is a fundamental database problem for many years. In addition, due to the emergence of the Web and the variety of data types, this problem has become more challenging. Query optimization techniques for both homogeneous and heterogeneous data are more and more needed.

The problem of query optimization is extensively investigated by many researchers, e.g., [GMUW08, RG02, AHV95]; and many optimization techniques have been incorporated in (the query optimizers of) conventional database systems. Most of these techniques translate each query to an optimal query execution plan (i.e., optimal order of physical operators). In particular, the conventional query optimizers examine multiple query execution plans for a single query and evaluate the most efficient one. Unlike the traditional single-query optimization, in multi-query optimization an optimal combined query execution plan for a collection of multiple queries is generated, e.g., [Sel88, PS88, RSSB00]. In particular, multi-query optimization techniques exploit commonalities between queries; for example by computing common subexpressions (i.e., subexpressions that are shared by multiple queries) once and reusing them, or by sharing scans of relations from disk. In this thesis we focus on higher-level optimization techniques which are based on transforming a collection of queries. Such transformations exploit sharing opportunities provided by a set of precomputed operations, and are taken place by focusing only on the definitions of queries.

Data collections are practically connected with many applications, which extract information by posing multiple queries, multiple times, on these collections. Moreover, in a data collection, there are base data units (such as base relations in relational databases) and several other data units that are resulted by queries over the base data. These additional data units are usually constructed either to provide a subset of information to an external application (e.g., for security purposes, or for improving the performance of data extraction) or to change the semantic meaning of some data of the collection (e.g., in information integration where semantic mappings are liable for integration). Such additional data units are presented by views and they can be either virtual or materialized (i.e., the results of the queries are stored in the disk).

In general, the views (either materialized or virtual) constitute a way to construct a middleware data collection. Namely, views can be used to reformulate the given collection. This reformulation is either automatically generated, using a specific set of constraints (view selection problem) or manually given (e.g., data integration, in which the information included in many related databases is joined into a whole). Such a reformulation is usually followed by an appropriate transformation of the queries posed on the initial data collection (e.g., in information integration, such a transformation is used for answering the queries posed on the mediated database). The technique of exploiting the data provided by a set of views in order to transform a set of queries (and consequently improve the performance of query evaluation) is the purpose of this thesis. This technique is also known as answering queries using views [Hal01].

The data collections may be conventional databases or other types of information, such as collections of Web data. The most important data model used by almost all commercial database systems is the relational model. The query language used by all relational databases is SQL (which stands for “Structured Query Language”). On the other hand, the most widely used data model for heterogeneous data is the semistructured-data model, of which XML (stands for “eXtensible Markup Language”) is the primary manifestation. In order, now, to query XML data, many query languages are proposed [ABS00, MB06]. Most of them use the XPath language to navigate through the XML data.

For SQL queries, three kinds of theoretical semantics have been proposed, the set, bag and bag-set semantics. In general, the best theoretical approximation of SQL semantics is the bag semantics because of its ability to manipulate multiple occurrences of the same tuple. However, in real database applications, the basic design principles do not recommend multiple occurrences of the same tuple in the base relations. More specifically, due to normalization rules, in every base relation there is a set of attributes, comprising the primary key of the relation, that uniquely identifies each tuple. Thus, every relation in a normalized relational database is regarded as a set. The best approach to handle theoretically the real needs is to focus on bag-set semantics, where the base relations are sets and the operators are liable for bag-results. In set semantics, no duplicate tuples allowed either in the base relations or the answers of queries.

In this thesis, we investigate techniques for query optimization using a set of views. In particular, for both relational and XML data collections, we focus on the problem of efficiently answering a set of queries using views. Moreover, the transformation needed in order to declare queries in terms of views (also called query rewriting) is achieved using query containment and equivalence [Hal01]; features that enable comparisons between results of queries. These features are also investigated for queries under realistic semantics (i.e., bag and bag-set semantics), in this thesis.

The query containment and equivalence problems have been extensively investigated for various types of database queries during the past two decades, but the focus is on relational databases, and considering set semantics. The feature of containment is based on checking only the forms of two queries and deciding whether or not the answer of one query is always (i.e., for every data collection) a part of the answer of the other query. The equivalence of two queries is given by two-way containment, and describes different reformulations of the same query. Here, we investigate the query containment problem for relational databases, considering either bag or bag-set semantics. In addition, we investigate the problems of containment and equivalence of unions of XPath queries.

Rewriting either a relational or a XPath query is also greatly investigated. However, the focus in relational databases is on set semantics. Here, we investigate the problem of rewriting a query using a set of views under bag and bag-set semantics, and describe the form of every rewriting of a given query using a specific set of views, together with the requirements that a set of views have to satisfy in order to give an equivalent rewriting of the given query. In the case of XML databases, we investigate the problem of rewriting a XPath query using multiple views; a problem which is not investigated in depth, in the past. In this case, we show that rewritings using multiple views may be required in order to get all the desirable information resulted by a given XPath query.

An efficient database reformulation, however, can be constructed by "automatically" designing the set of views. Especially, considering an initial set of conditions that the new data collection and the rewritings of the given queries have to satisfy (i.e., constraints over set of views and query rewritings), an optimal reformulation can be achieved. The constraints vary with respect to the application's needs. In the query optimization setting, the major constraint requires efficient query rewritings for the given set of queries. However, this is not practically the only constraint. For example, a limited storage space can also be added in the list of constraints. Namely, if we want to materialize the set of views, this set has to fit into the available storage space. In addition, since multiple updates are executed in the base data, a less-costly maintained set of views may be required. Considering, now, a given set of constraints, a data collection and a set of queries over this collection, the problem of finding together an efficient set of rewritings and an appropriate set of views for a given set of queries is called view selection. Here, we investigate for relational databases the view selection problem considering a storage limit constraint for the set of views, and finding the optimal query rewritings. Moreover, we succeed to solve the problem for both bag and bag-set semantics by describing the form of every possible query rewriting of a given query under bag and bag-set semantics, and by giving algorithms which outputs at least one optimal solution (when there exists a solution) for a given problem input.

1.1 Motivation scenarios

The problem of rewriting queries using views has recently received significant attention because of its relevance to a wide variety of data management problems: query optimization, information integration, data warehouse design and database security. Moreover, the view selection problem is also at the core of the query optimization, data warehousing and Web caching. In a data warehouse, a successful selection of views to materialize can preclude costly access to the base relations and consequently helps to answer a batch of queries in efficient way. In a similar manner, the choice of a proper set of views to precompute may improve the performance of Web-sites; because the set of expected queries can be answered quickly [FLSY99]. In information integration a set of views is used by each source database for determining the sharing data.

In information integration, a reformulation of the source databases is achieved by generating a new integrated database. This integrated database can be either physical (data warehouse) or virtual (mediated system). The integrated relations in both cases are views which are defined over the relations appearing in source databases. Especially, in the case of mediated systems, there are two approaches: the GAV (Global-As-View) and the LAV (Local-As-View) approach. In the first approach the views are defined in terms of source relations; instead of the second one, in which the source relations are defined in terms of views. The queries, now, posed in the integrated database have to be translated in terms of source relations. In the GAV approach this translation is straightforward; however, in the LAV approach rewriting a given query using a set of views is necessary. In data warehouses, since the amount of integrated information is huge, a view selection technique is used for query optimization.

In Web-site designs, precomputed views can be used to improve the performance of Web-sites [FLSY99]. Before choosing an optimal design, we must assure that the chosen views can be used to answer the expected queries at the Web-site. A system that caches answers locally at the client can avoid accesses to base relations at the server. Cached result of a query can be thought of as a materialized view, with the query as its view definition. The client could use the cached answers from previous queries to answer future queries. Therefore, the techniques of query rewriting and view selection can significantly help in an efficient Web-site designing.

In database security, security views are constructed in order to hide information [CFMS95]. Therefore, queries that are posed on the base database need to be transformed in terms of views. Hence, query rewriting and view selection techniques are used in order to efficient answer a given set of queries and designing such a set of views.

The following example describes a case in which the view selection is used on designing efficient Web-sites.

Example 1. Consider an airplane repair shop which stores information about its suppliers and its warehouses in three relations. The relational database has schema $\mathcal{S}$; and contains the relations $PART$, $WAREHOUSES$ and $SUPPLIERS$ with the following schemas (relation schemas):

$$PART(pName, wID)$$
$$WAREHOUSES(wID, wLocation, wSupervisor)$$
$$SUPPLIER(sName, sLocation)$$

In relation $PART$, each tuple says in which warehouse a certain part is stored; where the name of the part is given as a value of attribute $pName$ and the wID is a reference to a tuple included in the second relation. In relation $WAREHOUSES$, each record says where each of the shop's warehouses is located ($wLocation$), together with some other information like the supervisor of each warehouse ($wSupervisor$). Moreover, for each distinct warehouse there is a unique identifier given as a value of attribute wID. Finally, in the third relation, the suppliers ($sName$) and their locations ($sLocation$) are stored. In addition, we consider that in the relation

PART	
pName	wID
engine	1
engine	1
engine	2
wing	2
flap	1
flap	2

WAREHOUSES		
wID	wLocation	wSupervisor
1	Seattle	M.J.
2	Paris	J.L.

SUPPLIER	
sName	sLocation
Boeing	Seattle
Airbus	Paris

Figure 1.1: Database instance $\mathcal{D}$

PART multiple occurrences of tuples are allowed (i.e., both the relations of the database and the answers of the queries are multisets); which intuitively means that each warehouse may stores an amount of a certain part.

Let, now, a database instance $\mathcal{D}$ of the schema $\mathcal{S}$ which is illustrated in the Figure 1.1.

In addition, consider that the shop has placed their warehouses in such a way that the following happen: (1) each warehouse is located in a place that a supplier of the shop is located, and (2) each warehouse stores parts of the supplier located in the same place with the warehouse.

Considering, now, that an employee of the shop wants to find all parts contained in every warehouse of the shop, together with their suppliers, the query that is posed in the database may be the following (the query is given in two different query languages, *SQL* and *Datalog*).

$SQL : Q_1 :$

```
Select s.sName, p.pName
From SUPPLIERS s, WAREHOUSES w, PART p
Where s.sLocation=w.wLocation AND w.wID = p.wID;
```

$Datalog : Q_1 :$

```
q(X,Y) :-   SUPPLIER(X,Z), WAREHOUSES(W,Z,U),
            PART(Y,W)
```

The answer of the query Q_1 is illustrated as follows.

Q_1(D)	
sName	pName
Boeing	engine
Boeing	engine
Airbus	engine
Airbus	wing
Boeing	flap
Airbus	flap

We also consider that the following query is often posed in the database.

$SQL : Q_2 :$ **Select** p.pName, w.wLocation
From WAREHOUSES w, PART p
Where w.wID = p.wID;

$Datalog : Q_2 :$ q(Y,Z) :- WAREHOUSES(W,Z,U), PART(Y,W)

The query Q_2 asks for all parts the shop offers, together with the warehouses in which each part is stored. The answer of Q_2, now, when the query is posed on $\mathcal{D}$ is given as follows.

Q_2(D)	
pName	wLocation
engine	Seattle
engine	Seattle
engine	Paris
wing	Paris
flap	Seattle
flap	Paris

Let a Web-site that is connected to the above database, and extracts information through posing the queries Q_1 and Q_2 on $\mathcal{D}$, multiple times. In this case, one may think that every time the Web-site requests information from the database, the queries are posed and evaluated on $\mathcal{D}$. This technique, however, is not the most efficient one. This conclusion is easily implied by checking the definitions of the above queries. More specifically, we can easily notice that the join between the relations $WAREHOUSES$ and $PART$ is required during the evaluation of both Q_1 and Q_2. Therefore, precomputing this join operator and materializing the least amount of data required during the evaluation of both Q_1 and Q_2, we significantly speed up the response time of the Web-site. This materialization, however, requires the data resulted by the join to be fit into available disk space. Formally, now, we consider the views V_1 and V_2 with following definitions.

$V_1 : v_1(Y,Z)$:- $WAREHOUSES(W,Z,U), PART(Y,W)$.

$V_2 : v_2(X,Z)$:- $SUPPLIER(X,Z)$.

Notice that the first view describes the join we mentioned above. Moreover, notice that the first view is equal to the answer of Q_2 and the second view describes a copy of the relation $SUPPLIER$. Supposing, now, that the available disk space is equal to 8 tuples, we can materialize the view V_1. How can we, however, exploit the data of the views to answer the queries Q_1 and Q_2, more efficiently? The answer to this question is given by considering the following two queries that are defined in terms of V_1 and V_2.

$R_1 : r_1(X,Y)$:- $v_2(X,Z), v_1(Y,Z)$.

$R_2 : r_2(X,Y)$:- $v_1(X,Y)$.

Focusing on the definitions of the queries Q_1, Q_2, R_1 and R_2 and the definitions of the above views, we easily conclude that the answers of Q_1 and Q_2 are equal to the answers of R_1 and R_2, respectively. The queries R_1 and R_2 are called equivalent rewritings of Q_1 and Q_2, respectively, using the views V_1 and V_2. In this case, using R_1 and R_2 instead of Q_1 and Q_2, we avoid to compute the join between the relations $WAREHOUSES$ and $PART$, multiple times; hence we speed up query evaluation.

Suppose, now, that the available disk space is 12 tuples. In this case, we can materialize the answers of both Q_1 and Q_2 instead of using the views V_1, V_2 and the rewritings R_1 and R_2; a plan which speed up even more the extraction of the desirable information (i.e., the answers of Q_1 and Q_2). Here, we have to notice that in the case that the available disk space is equal to 8 tuples there is no set of (conjunctive) views that gives more efficient rewritings of Q_1 and Q_2. Considering, however, storage limit equal to 12 tuples, the most efficient rewritings of Q_1 and Q_2 are defined as follows:

$R'_1 : r'_1(X,Y)$:- $v'_1(X,Y)$,

$R'_2 : r'_2(X,Y)$:- $v'_2(X,Y)$,

where the definitions of the materialized views V'_1 and V'_2 are the following.

$V'_1 :\ v'_1(X,Y)$:- $SUPPLIER(X,Z), WAREHOUSES(W,Z,U), PART(Y,W)$,

$V'_2 :\ v'_2(Y,Z)$:- $WAREHOUSES(W,Z,U), PART(Y,W)$.

□

1.1.1 Organization of thesis

This thesis is organized as follows.

- In the Chapter 2, the preliminaries for querying relational databases using views are given. In this chapter, we describe the basic concepts and techniques for deciding query containment and equivalence. In addition, the problems of rewriting queries using views and view selection are described using a detailed analysis of the related work.

- In the Chapter 3 [ADG09], we investigate the problem of query containment under bag and bag-set semantics through a detailed analysis of special cases. The complexity in each case is given, as well.

- In the Chapter 4 [ADG08, ADGa], the problem of view selection under bag and bag-set semantics is investigated. In this chapter, we describe the form of every query rewriting of a given query, considering bag-set semantics. Moreover, we give a necessary and sufficient condition (over a set of views) for finding an equivalent rewriting of a given query; considering relational database and bag-set semantics.

- In the Chapter 5, the basic concepts and techniques for rewriting XPath queries using views, and preliminaries for XML databases, are given. The problems of containment and equivalence of XPath queries is also given through a detailed analysis of the related work.

- In the Chapter 6 [ADGb], the problems of rewriting XPath patterns using multiple views and the containment and equivalence of unions of XPath queries are investigated.

Chapter 2

Querying relational databases using views

Nowadays, the relational databases is by far the dominant type of databases. In this chapter, we focus on the relational databases and we especially describe the basic concepts and techniques of answering queries using views and selecting views with respect to a given set of queries over a specific relational database schema. In addition, we focus on the class of conjunctive queries which is one of the most interesting classes of SQL queries and the class that has been greatly investigated.

The problems of containment and equivalence of conjunctive queries are also described for conjunctive queries, through a detailed analysis of techniques used for deciding these problems. In addition, both problems are analyzed for each of the three theoretical approximations of the SQL semantics; the set, the bag and the bag-set semantics.

2.1 Basic definitions

We consider a collection of finite elements A. We say that A is a *set* if every element in A is distinct. Supposing that there is at least one element e that appears n times in the collection A, where $n \geq 1$, we say that A is a *bag* (or *multiset*). The number n is referred as the *multiplicity* of e in the bag A. Notice that every set is also a bag considering that every element in the set has multiplicity equal to 1. A bag can be also thought as a set of pairs $(e; n)$, where e is an element in the bag and n represents the multiplicity of e. Throughout this thesis, we use the above collapsed notation to represent bags; instead of listing all occurrences of each element.

Moreover, for shorthand, we omit to write the multiplicity of an element that appears once in a bag.

A relation is a named, two-dimensional table of data. Each column of the relation has a distinct name that is called *attribute*; and each row of data (called *tuple*) is sequence of values on the attributes of the relation, one value for each attribute. The number of the attributes is called *arity* of the relation. A *relation schema* is a signature of the form $r(A_1, A_2, \ldots, A_n)$ defined by the name r of the relation (called *relation name*) and the sequence of attributes $(A_1, A_2, \ldots, A_n)$. Moreover, a collection of tuples over a relation schema is called *relation instance*. A relation can be viewed either as a set or as a bag of tuples (also called *bag-relation*). A (*relational*) *database schema* is a set of relations schemas. A (*relational*) *database instance* (*database*, for short) of schema $\mathcal{S}$ is a set of relations instances of schemas in $\mathcal{S}$. Throughout this thesis, we describe a database instance by listing the tuples appearing in each relation instance of a database, and attaching to each tuple the name of the relation the tuple appears. For example, the database $\mathcal{D} = \{r_1(a, b), r_2(c, d)\}$ contains two tuples; the tuple (a, b) included in the relation r_1 and the tuple (c, d) included in the relation r_2. In addition, we denote as $r(\mathcal{D})$ the instance of the relation r in a database $\mathcal{D}$. We consider a database either as *set-valued*, if all stored relations are sets, or as *bag-valued*, if multiset stored relations are allowed.

Considering a bag-valued database $\mathcal{D}$, we define the *set-valued representation* of $\mathcal{D}$, denoted as $set(\mathcal{D})$, to be the set-valued database instance that is produced by eliminating duplicates from relations of $\mathcal{D}$.

2.1.1 Operators

Since the conventional set-operators (e.g. Union, Cartesian Product, Intersection, e.t.c.) have not the ability to manipulate duplicate elements, a similar set of operators, called *bag-operators* [GMUW08], is introduced. The bag-operators can be thought as generalizations of set-operators such that the duplicate elements are taken account. More specifically, in bag-operators all occurrences of a certain element are treated as distinct elements. In the following, we differentiate the bag-operators from set-operators by subscripting each operator with the letter b; instead of set semantics where we subscript each operator with the letter s.

Generalizing the subset operator, denoted as $\subseteq_s$, in order to handle bags, we say that a bag B_1 is a *subbag* of a bag B_2, denoted as $B_1 \subseteq_b B_2$, if every element $e \in B_1$ with multiplicity n_1, is also contained in B_2 with multiplicity n_2, where $n_1 \leq n_2$. Similarly, we say that two bags B_1 and B_2 are *equal*, denoted $B_1 =_b B_2$, if $B_1 \subseteq_b B_2$ and $B_2 \subseteq_b B_1$.

The Cartesian product of bags is analogously defined as the Cartesian product of sets. More specifically, to compute the *Cartesian product* $B_1 \times_b B_2$ of two bags B_1 and B_2, each element of B_1 is paired with each element of B_2, regardless of whether it is a duplicate or not. As a result, if an element e_1 appears n_1 times in B_1, and an element e_2 appears n_2 times in B_2, then the element (e_1, e_2) will appear $n_1 \cdot n_2$ times in $B_1 \times_b B_2$.

When we take the *union* of two bags, we add the number of occurrences of each element. That is, if B_1 is a bag in which the element e appears n_1 times, and B_2 is a bag in which the element e appears n_2 times, then in the bag $B_1 \cup_b B_2$ the element e appears $n_1 + n_2$ times. When we *intersect* two bags B_1 and B_2, in which element e appears n_1 and n_2 times, respectively, in $B_1 \cap_b B_2$ the element e appears $min(n_1, n_2)$ times. When we compute the *difference* $B_1 -_b B_2$ of bags B_1 and B_2, the element e appears in $B_1 -_b B_2$ $max(0, n_1 - n_2)$ times. That is, if e appears in B_1 more times than it appears in B_2, then in $B_1 -_b B_2$ the element e appears the number of times it appears in B_1, minus the number of times it appears in B_2. However, if e appears at least as many times in B_2 as it appears in B_1, then e does not appear at all in $B_1 -_b B_2$. Intuitively, each occurrence of e in B_2 "cancels" one occurrence in B_1.

Concerning the *selection* operator applied to a bag-relation, the selection condition is applied on each occurrence of every tuple independently. Thus its effect is the same as the effect of conventional selection. *Projection* operator on a bag is also similar to projection on a set as it is applied on each occurrence of every tuple independently. However, the duplicates are not eliminated in the result of the projection applied on a bag. In addition, in the case that the elimination of some attributes causes the same tuple t to be obtained from two or more different tuples, the multiplicity of t in the result of the projection is the sum of the multiplicities of the initial tuples.

Joining bag-relations also present no surprises. We compare each occurrence of every tuple of one relation with each occurrence of every tuple of the other, decide whether or not this pair of tuples joins successfully, and if so we put the resulting tuple in the answer. Constructing the result, we do not eliminate duplicate tuples. In addition, it is important to note that joining a relation R with itself (such that all the corresponding attributes are equal) using bag-operators does not result the relation R (i.e. $R \bowtie_b R \neq_b R$); instead of the case of set-operators where this property holds (i.e. $R \bowtie_s R =_s R$). This notice implies, as we will see in the following, that in the case of bag-valued databases, joining a relation with itself multiple times affect, significantly, the result of a sequence of operators.

Example 2. Let a database schema $\mathcal{S}$ that contains the binary relations *edge* and *path* such that the first relation stores the edges of a directed graph, and the second one, its paths of length 2. Also consider that we store in the database the directed graph G depicted in Figure 2.1. Moreover, we

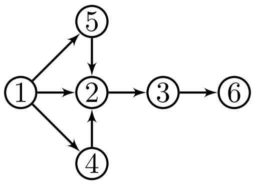


Figure 2.1: Directed Graph G

notice that the bag-valued database instance $\mathcal{D}$ of $\mathcal{S}$ (in which G is stored) contains the relation instances: $edge(\mathcal{D}) = \{(1,2), (1,4), (1,5), (2,3), (3,6), (4,2), (5,2)\}$ and $path(\mathcal{D}) = \{((1,2);2), (1,3), (2,6), (5,3), (4,3)\}$. It is easy to see that the tuple $(1,2)$ appears once, both in $edge(\mathcal{D}) \cap_b path(\mathcal{D})$ and in $path(\mathcal{D}) -_b edge(\mathcal{D})$, none in $edge(\mathcal{D}) -_b path(\mathcal{D})$, and three times in $edge(\mathcal{D}) \cup_b path(\mathcal{D})$. In addition, selecting the paths of length 2 starting from the node labeled by 1 (i.e. select the tuples from $path(\mathcal{D})$ that have the value 1 in the first attribute) we get the bag $\{((1,2);2), (1,3)\}$. Projecting, using bag-operators, on the second attribute of the (set-)relation $edge(\mathcal{D})$ it is important to note that the result is the bag $\{(2;3),\ 3,\ 4,\ 5,\ 6\}$. Searching for pairs of nodes that there is a path of length 4 between them, we join the bag-relation $path(\mathcal{D})$ with itself such that the second attribute of the first is equal to the first attribute of the second. It is easy to see that in the resulting bag the tuple (1,2,6) appears two times. Moreover, selecting from relations $edge(\mathcal{D})$ and $path(\mathcal{D})$ the tuple $(1,2)$, we have that for the resulting bags B_{edge}, B_{path}, respectively, it holds that $B_{edge} \subseteq_b B_{path}$ but $B_{edge} \neq_b B_{path}$ (because the tuple (1,2) appears more times in B_{path} than B_{edge}). □

2.1.2 Expressions and substitutions

The operators over sets and bags provide a way to manage the information stored in collections of data. In this perspective, we use the concept of expression to denote an arbitrary sequence of joins, Cartesian products and selections over relations. More specifically, an *expression* is a conjunction of atoms of the form $g_1(\overline{X}_1)$, $g_2(\overline{X}_2)$, $\ldots$, $g_n(\overline{X}_n)$, where each *atom* $g_i(\overline{X}_i)$, with $i = 1, \ldots, n$, is a reference to the relation g_i, and $\overline{X}_i$ is a vector of variables and constants which replace the attributes in the relation schema of g_i. Moreover, we say that an expression E has *self-joins* if it has more than one atoms with the same relation name. Throughout this thesis, we use capital letters to denote variables of an expression and small letters to denote constants (except if explicitly mentioned).

The physical meaning of an expression is given as follows. Multiple occurrences of a variable in an atom and the existence of constants inside an atom indicate use of selection operator (i.e. select from a certain relation the tuples such that the values of two or more attributes are equal or, respectively, select tuples with specific values in corresponding attributes). In addition, the occurrence of a variable in two different atoms of an expression indicates join with equality comparison; and if two atoms have no common variables, then the operator that is applied on the corresponding relations is the Cartesian product. The use of the expression, however, does not provide any information about the ordering of the specific operators, in general. Here, we consider that the selections are pushed down in the evaluation plan (we firstly apply selections on relations and then we apply joins and Cartesian products, in arbitrary order) [GMUW08]. Moreover, depending on which semantics we consider, the result of an expression is given by computing the corresponding operators.

Example 3. Let the relations *employee* and *department* with the following schemas:
$employee(Emp\_ID, First\_Name, Father\_Name, Last\_Name, Dept\_ID)$
$department(Dept\_ID, Department\_Name, Manager)$

Now, consider the following expressions over the above relations:

$E_1 = employee(X, john, Y, Z, W), employee(A, Y, B, C, W)$
$E_2 = employee(U, Y, Y, V, W), department(A, B, C)$

The expressions E_1, E_2 indicate that the following operators are applied on the above relations. As we can see, E_1 implies a selection over the relation *employee* searching for tuples that have the value *john* in the attribute $First\_Name$. Moreover, E_1 joins the relation *employee* to itself searching for employees that work in the same department; and the father's name of the first is the first name of the second. The expression E_2 selects every employee with same first name and father's name; and also apply a Cartesian product between the relations *employee* and *department*. □

The result of an expression is either a bag or a set with respect to the operators we consider. We denote as $E(\mathcal{D})$ the resulting collection (bag or set, respectively) of an expression E when E is applied on a database $\mathcal{D}$ (bag-valued or set-valued, respectively). Since the result of an expression

E is either a bag- or a set-relation, we may apply additional operators on the resulting collection of E. In the following definition, we formally define the concept of substitution; which intuitively is a way to construct an expression E' from an expression E in such a way that for every database $\mathcal{D}$ the resulting collection $E'(\mathcal{D})$ can be produced by applying a sequence of additional selections on $E(\mathcal{D})$.

Definition 1. A **substitution** θ is a finite set of the form $\{X_1/Y_1, \ldots, X_n/Y_n\}$, where each Y_i is a variable or a constant, and $X_1, \ldots, X_n$ are distinct variables. When $Y_1, \ldots, Y_n$ are distinct variables, θ is called **renaming substitution**. Let $\theta = \{X_1/Y_1, \ldots, X_n/Y_n\}$ be a substitution. Then the **instance** $\theta(E)$ of an expression E, is the expression obtained by simultaneously replacing each occurrence of X_i in E by Y_i for all $i = 1, \ldots, n$; while the expression E is called a **generalization** of the expression $E' = \theta(E)$.

Also, we denote as $\theta(X) = Y$ that the variable X is substituted by the variable Y using the substitution θ. Here, we have to notice that applying a substitution θ on an expression E, denoted $\theta(E)$, each variable of E is substituted using θ, even if it is substituted by itself; i.e. all the variables of E are in the domain of θ.

Considering two expressions E_1 and E_2, we say that a substitution θ over E_1 is a *mapping* from E_1 to E_2, denoted as $\theta : E_1 \rightarrow E_2$, if the expression produced by eliminating duplicate atoms from $\theta(E_1)$ is a subexpression of E_2. In addition, an atom g of E_1 *maps* on an atom g' of E_2 w.r.t. θ if $g' = \theta(g)$; and similarly, we say that a variable X of E_1 *maps* on the variable Y of E_2 w.r.t. θ if $Y = \theta(X)$.

Moreover, the definitions of expression and substitution imply that each instance of an expression E describes a sequence of additional selections over the result of E; i.e. considering a substitution θ over E the result of $\theta(E)$ is produced by applying a sequence of selections on the result of E. The following example illustrates this remark.

Example 4. Let an instance $\mathcal{D}$ of the database schema $\mathcal{S}$ illustrated in the Example 3, such that $\mathcal{D}$ contains the following relation instances:

$$employee(\mathcal{D}) = \{(1, John, John, Smith, 1), (2, Mat, George, Johnson, 2)\}$$
$$department(\mathcal{D}) = \{(1, IT, 1), (2, Marketing, 2)\}$$

The result $E_2(\mathcal{D})$ of applying the expression E_2, illustrated in the Example 3, on the database $\mathcal{D}$ using set-operators (it is the same using bag-operators) is:

$$E_2(\mathcal{D}) = \{(1, John, John, Smith, 1, 1, IT, 1), (1, John, John, Smith, 1, 2, Marketing, 2)\}.$$

Selecting from the relation $E_2(\mathcal{D})$ the tuples such that each employee works in the department with id equal to the id of the employee, it is easy to see that the resulting relation contains only the tuple (1, *John*, *John*, *Smith*, 1, 1, *IT*, 1). It is easy to verify that this result is also the result of the expression:

$$E_3 = employee(U', Y', Y', V', U'), department(U', B', C')$$

when we apply E_3 to the database $\mathcal{D}$. Notice that E_3 is the result of applying the substitution $\theta = \{U/U', Y/Y', V/V', W/U', A/U', B/B', C/C'\}$ to the expression E_2; i.e. E_3 is a generalization of E_2 (or E_2 is an instance of E_3). □

Notice, here, that each expression may have multiple instances and consequently, multiple generalizations. Therefore, considering a set of expressions $\mathcal{E}$ there may be expressions that generalize each expression of $\mathcal{E}$; i.e. there may be *common generalization* of the expressions in $\mathcal{E}$. In addition, if there is at least one common generalizations of expressions in $\mathcal{E}$ we define the concept of *least general generalization* [Plo70] which is a common generalization that requires the least number of selections in order to become identical (up to a renaming substitution) to each expression of the set $\mathcal{E}$.

Definition 2. An expression E is a **common generalization** of $E_1, \dots, E_n$, with $n > 1$ if E is a generalization of each expression E_i, with $1 \le i \le n$. E is a **least common generalization** (or a **least general generalization -lgg**) of $E_1, \dots, E_n$, with $n > 1$, if E is a common generalization of $E_1, \dots, E_n$, and there is no other common generalization G of $E_1, \dots, E_n$, such that E is a generalization of G.

Example 5. Let the following five expressions posed on the binary relations r_1 and r_2:

$$E_1 = r_1(Y, X), r_1(X, X), r_2(X, b), r_2(a, a)$$
$$E_2 = r_1(Y, X), r_1(X, X), r_2(X, a), r_2(a, b)$$
$$E_3 = r_1(D, A), r_1(A, A), r_2(A, B), r_2(a, C)$$
$$E_4 = r_1(D, A), r_1(A, A), r_2(B, a), r_2(C, b)$$
$$E_5 = r_1(D', A'), r_1(A', A'), r_2(A', B'), r_2(C', D')$$

Notice that neither E_2 is generalization of E_1 nor E_1 is generalization of E_2. The expression E_3 is a common generalization of E_1 and E_2, because applying the substitutions $\theta_{31} = \{Y/D,\ A/X\ B/b,\ C/a\}$ and $\theta_{32} = \{Y/D,\ A/X,\ B/a,\ C/b\}$ on E_3 we get the expressions E_1 and E_2, respectively. Similarly, we can verify that E_4 and E_5 are also common generalizations of E_1 and E_2. In addition, it is easy to see that E_3 and E_4 are two different lggs of E_1 and E_2. On the other hand, the expression E_5 is not an lgg of E_1

and E_2, because applying the substitution $\theta = \{D'/D,\ A'/A,\ B'/B,\ C'/a,\ D'/C\}$ on E_5, we get the expression E_3. $\square$

On constructing least general generalization

The definitions of common generalization and lgg of a set of expressions imply a simple condition on the form of the expressions. This condition is represented in the following remark, and it is easily implied by the definition of substitution.

Remark 1. Let the set of expressions Exp. Then there is at least one lgg of the expressions in Exp if and only if for each relation name r appearing in an expression in Exp there is the same number of atoms with relation name r in every expression in Exp.

Let, now, the set of n expressions $Exp = \{E_1, E_2, \ldots, E_n\}$ such that the expressions in Exp satisfy the condition described by Remark 1. Also, consider that each expression in Exp has k atoms. Moreover, we suppose an ordering of the atoms of each expression such that $g_{(i,j)}$ is the i-th atom of the j-th expression. Each lgg of the expressions in Exp is inductively constructed using the function h_{lgg} which is defined as follows [BG95]:

- $h(Z_1, \ldots, Z_n) = X$, where X is either a constant if for each $i = 1, \ldots, n$ we have that Z_i is identical to X, or otherwise a fresh variable,
- $h_{lgg}(e(Z_{11}, \ldots, Z_{1m}), \ldots, e(Z_{n1}, \ldots, Z_{nm}))) = e(h_{lgg}(Z_{11}, \ldots, Z_{n1}), \ldots, h_{lgg}(Z_{1m}, \ldots, Z_{nm}))$, where e is a relation of arity m,
- $h_{lgg}(E_1, \ldots, E_n) = h_{lgg}(g_{(1,1)}, \ldots, g_{(1,n)}), \ldots, h_{lgg}(g_{(k,1)}, \ldots, g_{(k,n)})$.

It is easy to notice, now, that using different orderings of the atoms of expressions in Exp we can construct different lggs. In addition, this technique does not always result an lgg. Namely, using every combination of orderings of the atoms of the expressions, we get a set of common generalizations of the expressions in Exp. This set, however, contains every lgg of the given expressions.

Example 6. Let the expressions described in Example 6. Moreover, consider an ordering of the atoms of E_1 and E_2 described as follows

$$\begin{tikzcd}
E_1 = & r_1(Y,X) \arrow[d, dash] & r_1(X,X), \arrow[d, dash] & r_2(X,b), \arrow[d, dash] & r_2(a,a) \arrow[d, dash] \\
E_2 = \arrow[d, phantom, "h_{lgg}"] & r_1(Y,X) \arrow[d] & r_1(X,X), \arrow[d] & r_2(X,a), \arrow[d] & r_2(a,b) \arrow[d] \\
E_3 = & r_1(D,A) & r_1(A,A), & r_2(A,B), & r_2(a,C)
\end{tikzcd}$$

The construction of the lgg using the function h_{lgg} is also illustrated above (we remind that a, b and c are constants). Swaping, now, the first two atoms of E_2 (i.e., we use a different ordering), the expression E_6, that is also constructed using the above technique, is the following.

$$\begin{array}{rcccc}
E_1 = & r_1(Y,X) & r_1(X,X), & r_2(X,b), & r_2(a,a) \\
 & | & | & | & | \\
E_2 = & r_1(X,X) & r_1(Y,X), & r_2(X,a), & r_2(a,b) \\
h_{lgg} & \downarrow & \downarrow & \downarrow & \downarrow \\
E_6 = & r_1(D,A) & r_1(E,A), & r_2(A,B), & r_2(a,C)
\end{array}$$

Notice, however, that E_6 is a common generalization of E_1 and E_2 and a generalization of E_3. Thus E_6 is not an lgg of E_1 and E_2. □

2.2 Queries and Views

A *query* is a mapping from databases to databases, usually specified by a logical formula on the schema $\mathcal{S}$ of the input databases. Typically, the output database (called *query answer*) is a database with a single relation. In this thesis we focus on the class of select-project-join SQL queries (SPJ queries, for short) with equality comparisons, a.k.a. *conjunctive queries* (CQs for short). Formally, the definition of a conjunctive query [AHV95] is a rule of the form:

$$Q : q(\overline{X}) \text{ :- } g_1(\overline{X}_1), \ldots, g_n(\overline{X}_n)$$

where the expression on the right of :- is applied on the (no necessarily distinct) relations $g_1, \ldots, g_n$, and the relation name q is a fresh relation. The atom $q(\overline{X})$ is the *head* of Q, denoted *head*(Q) while the expression on the right of :- is said to be the *body* of Q, denoted *body*(Q). Each $g_i(\overline{X}_i)$, with $i = 1, \ldots, n$, is also called a *subgoal* of Q. The variables in $\overline{X}$ are called *distinguished* or *head variables* of Q and the set of distinguished variables of Q is denoted as $DVars(Q)$, whereas the variables in $\overline{X}_i$, $i = 1, \ldots, n$, are called *body variables* of Q and the set of body variables of Q is denoted as $BodyVars(Q)$. In addition, we denote the set of variables of an atom $g(\overline{Z})$ as $vars(g(\overline{Z}))$ (or as $vars(\overline{Z})$, for short). A body variable which is not also a head variable is called *non-distinguished variable* of Q and the set of non-distinguished variables of Q is denoted as $NDVars(Q)$. Moreover, we say that a CQ has *self-joins* if its body has self-joins. In this thesis, we consider *safe conjunctive queries* that is CQs whose head variables also occur in their body.

Remark 2. For every safe CQ Q we have that $DVars(Q) \cup NDVars(Q) = BodyVars(Q)$.

The existence of variables in the head of a safe CQ Q indicates use of the projection operator to the result of applying the body of Q on a database instance; i.e., we project to the attributes of the body the distinguished variables appear.

The following example illustrates the correspondence between SPJ queries and a CQ written as logical formula.

Example 7. Let the database schema $\mathcal{S}$ introduced in the Example 3. Searching for the managers of the departments which contain an employee who has the same first name and father's name and his last name is *Smith*, we pose on the database the following query, written using SQL syntax:

```
select d.Manager
from employee e, department d
where e.First_Name = e.Father_Name AND
e.Dept_ID=d.Dept_ID AND e.Last_Name='Smith'
```

The logical formula that equivalently represents the above SQL query is the following:

$$Q: \ q(X) \text{ :- } employee(Y, Z, Z, Smith, W), department(W, V, X)$$

It is easy to see that the projection operator over the attribute $Manager$ is represented by exporting only the variable X to the head of the definition of CQ (notice that X is appeared in the position of the relation department that corresponds to the attribute $Manager$). □

Since there is no information about the ordering of the operators in the query plan in the above logical form of CQs, we consider use of only *left-linear query plans*, where selections are pushed as far as they go and projection is the last operator.

In addition, as a CQ is constituted by two expressions (the head and the body) we extend the concepts of instance and generalization of the expressions to CQs. Therefore, applying the same substitution θ both to the head and to the body of a CQ Q (we say that θ is applied on Q), the CQ $\theta(Q)$ is an *instance* of Q; while we say that Q is a *generalization* of the $\theta(Q)$. A CQ Q is a *common generalization* of a set of queries $\mathcal{Q}$ if Q is a generalization of each query in $\mathcal{Q}$. Moreover, Q is a *least common* (or *general*) *generalization* (*lgg*, for short) of a set of CQs $\mathcal{Q}$ if Q is a common generalization of the queries in $\mathcal{Q}$ and there is no other common generalization Q' of $\mathcal{Q}$ such that Q is a generalization of Q'.

Moreover, we refer to a *view* as a named query. A view is said to be *materialized* if its answer is stored in the database. In this thesis, we are restricted to the use of views defined by conjunctive queries called *conjunctive views* (except if explicitly mentioned).

2.2.1 Semantics of Queries

In this section we give the semantics of a CQ (written as a logical formula). In the following we use the concept of a *valuation* [AHV95] from a database instance $\mathcal{D}$ on a CQ Q; which is a substitution over Q such that each variable of Q is replaced by a constant, and the facts that are produced in the body of Q are tuples in $\mathcal{D}$. In addition, considering a database schema $\mathcal{S}$, a bag-valued database instance $\mathcal{D}$ of $\mathcal{S}$, a query Q over $\mathcal{S}$ and a valuation θ from $\mathcal{D}$ on Q, we denote as *multiplicity of the instance* $\theta(Q)$ (or the multiplicity of θ) the multiplication of the multiplicities of the tuples in $\mathcal{D}$ that also appear in the body of $\theta(Q)$.

Considering, now, a CQ Q over a database schema $\mathcal{S}$ and a database instance $\mathcal{D}$ of $\mathcal{S}$, each tuple t produced in the head of Q after applying a valuation θ on Q, i.e. $t = \theta(head(Q))$, appears in the answer $Q(\mathcal{D})$. Moreover, for each tuple in the answer of a query Q when Q is posed on a database instance $\mathcal{D}$, t corresponds to a valuation from $\mathcal{D}$ on Q. Formally, considering a database schema $\mathcal{S}$, a CQ over $\mathcal{S}$ and an instance $\mathcal{D}$ of $\mathcal{S}$, the answer $Q(\mathcal{D})$ of Q is the following relation:

$$Q(\mathcal{D}) = \{t | t = \theta(head(Q)) \text{ such that } \theta \text{ is a valuation from } \mathcal{D} \text{ on } Q\}.$$

In addition, supposing a set-valued database instance we may consider the CQ answer either as a set- or as bag-relation; however, in the case of bag-valued databases the answer of a CQ answer is a bag-relation. Considering set-valued database instance and set-operators we evaluate a CQ under *set semantics*; considering set-valued database instance and bag-operators we evaluate a CQ under *bag-set semantics*, and considering bag-valued database instance and bag-operators we evaluate a CQ under *bag semantics*.

Remark 3. Let a CQ Q over a database schema $\mathcal{S}$. Then the following hold:

- For every set-valued database instance $\mathcal{D}$ of $\mathcal{S}$, evaluating Q under set semantics we have that for every tuple t in $Q(\mathcal{D})$ there exists at least one valuation θ from $\mathcal{D}$ on Q such that $t = \theta(head(Q))$.
- For every set-valued database instance $\mathcal{D}$ of $\mathcal{S}$, evaluating Q under bag-set semantics we have that for every tuple t with multiplicity n in

$Q(\mathcal{D})$ there exists a set Θ of n distinct valuations from $\mathcal{D}$ on Q such that for each $\theta \in \Theta$ it holds that $t = \theta(head(Q))$.

- For every bag-valued database instance $\mathcal{D}$ of $\mathcal{S}$, evaluating Q under bag semantics we have that for every tuple t with multiplicity n in $Q(\mathcal{D})$ there exists a set Θ of distinct valuations from $\mathcal{D}$ on Q such that for each $\theta \in \Theta$ it holds that $t = \theta(head(Q))$ and the sum of the multiplicities of the valuations in Θ is equal to n.

Example 8. Let the database schema $\mathcal{S}$ that contains the binary relation *edge*. Also consider the set-valued database instance $\mathcal{D}$ which contains the relation instance $edge(\mathcal{D})$ given by the graph G illustrated in Figure 2.2.

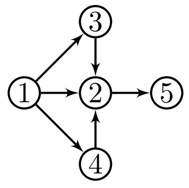


Figure 2.2: Directed Graph G

Consider, now, the query *path* with definition:

$$path:\ path(X,Y) \text{ :- } edge(X,Z), edge(Z,Y)$$

that results the paths of length 2 appearing on a directed graph.

Evaluating the query *path* on $\mathcal{D}$ under set semantics, there are five different valuations from $\mathcal{D}$ on *path*; two of them give the tuple $t = (1,2)$ in $head(path)$ but t appears only once in $path(\mathcal{D})$. More specifically, the first valuation θ_1 from $\mathcal{D}$ on *path* is $\{X/1, Z/3, Y/2\}$, the second one, named θ_2, is $\{X/1, Z/4, Y/2\}$, and the other three, named θ_3, θ_4 and θ_5, substitute X with 1, 3 and 4, respectively, and the variable Z with 2 and Y with 3. Hence, $path(\mathcal{D}) = \{(1,2), (1,5), (3,5), (4,5)\}$.

Supposing bag-set semantics for the evaluation of the query, θ_1 and θ_2 are two distinct valuations from $\mathcal{D}$ on *path* which give two occurrences of the same tuple in $path(\mathcal{D})$; hence $path(\mathcal{D}) = \{((1,2);2), (1,5), (3,5), (4,5)\}$.

Now consider the database with schema $\mathcal{S}'$ that contains the relation *path* described above. Also consider the instance $\mathcal{D}'$ of $\mathcal{S}'$ such that $path(\mathcal{D}') = path(\mathcal{D})$. Suppose the query Q over the schema $\mathcal{S}'$ with definition:

$$Q:\ q(X) \text{ :- } path(X,Y), path(X,Z)$$

that finds the nodes from which two paths of length 2 in a directed graph start. Moreover, as the database is bag-valued, we evaluate Q under bag semantics. As we can see, there are four distinct valuations that give the node 1 in $head(Q)$; these valuations are: θ'_1, θ'_2, θ'_3, θ'_4 from $\mathcal{D}'$ on Q, where $\theta'_1 = \{X/1, Y/2, Z/2\}$, $\theta'_2 = \{X/1, Y/2, Z/5\}$, $\theta'_3 = \{X/1, Y/5, Z/2\}$, and $\theta'_4 = \{X/1, Y/5, Z/5\}$. Moreover, the multiplicities of the above valuations are $n_1 = 4$, $n_2 = 2$, $n_3 = 2$ and $n_4 = 1$, respectively. Hence, $q(\mathcal{D}') = \{(1; 9), (3), (4)\}$. □

Corollary 1 is immediately implied from the definitions of bag-operators, and describes the correspondence of the evaluations of CQs under set, bag and bag-set semantics.

Corollary 1. *Let Q_1 be a CQ under set semantics, Q_2 be a CQ under bag-set semantics and Q_3 be a CQ under bag semantics such that Q_1, Q_2 and Q_3 are syntactically the same query. For every bag-valued database instance $\mathcal{D}$ the following hold:*

1. $Q_1(set(\mathcal{D})) =_s set(Q_2(set(\mathcal{D}))) =_s set(Q_3(\mathcal{D}))$*; and*
2. $Q_1(set(\mathcal{D})) \subseteq_b Q_2(set(\mathcal{D})) \subseteq_b Q_3(\mathcal{D})$.

Practically, the set semantics corresponds to SQL semantics supposing that the answer of each SPJ query is produced by the necessarily use of the keyword DISTINCT. On the other hand, bag-set and bag semantics correspond to SQL semantics considering no use of DISTINCT keyword. The bag-set semantics indicates that the database instance is set-valued; while under bag semantics the database instance is bag-valued. In practice, a set-valued database is usually achieved by using basic relational database designing rules (a.k.a. normalization rules).

2.3 Containment and Equivalence

In this section, we focus on the notions of query containment and query equivalence; features of queries that are used by the conventional relational database systems and extensively investigated by many database research areas, such as query optimization, view selection and information integration. Query containment and query equivalence enable comparison between different reformulations of queries, in such a way that the answer of a query is guaranteed to be either a part of or equal to, respectively, the answer of an another query. The formal definitions of the query containment and query equivalence are given as follows.

Definition 3. Let Q_1 and Q_2 be two queries over a database schema $\mathcal{S}$ under set semantics (bag semantics, bag-set semantics, respectively). We say that Q_2 is **set contained** (**bag contained**, **bag-set contained**, respectively) in Q_1, denoted $Q_2 \sqsubseteq_s Q_1$ ($Q_2 \sqsubseteq_b Q_1$, $Q_2 \sqsubseteq_{bs} Q_1$, respectively), if for every set-valued (bag-valued, set-valued, respectively) database instance $\mathcal{D}$ of $\mathcal{S}$, we have that $Q_2(\mathcal{D}) \subset_s Q_1(\mathcal{D})$ ($Q_2(\mathcal{D}) \subseteq_b Q_1(\mathcal{D})$, $Q_2(\mathcal{D}) \subseteq_b Q_1(\mathcal{D})$, respectively). **Equivalence** for each semantics is defined as two way containment.

The following example illustrates a simple case of query optimization which is based on the concepts of query containment and query equivalence.

Example 9. Consider the database schema $\mathcal{S}$ introduced in the Example 3 and a set-valued database instance $\mathcal{D}$ of $\mathcal{S}$. Also, consider two materialized views V_1, V_2 over $\mathcal{S}$ with definitions:

$$V_1 : v_1(X, Z, Y) \text{ :- } \mathit{employee}(X, Z, Y, \mathit{Smith}, W)$$
$$V_2 : v_2(X, Y, Y) \text{ :- } \mathit{employee}(X, Y, Y, \mathit{Smith}, W).$$

V_1 stores the employees whose last name is *Smith*, and V_2 stores the employees whose first name is the same with their father's name and their last name is *Smith*. Suppose the query Q over $\mathcal{S}$, defined by the following rule:

$$Q : q(X, Y, Y) \text{ :- } \mathit{employee}(X, Y, Y, \mathit{Smith}, W)$$

Q searches for employees whose first name is the same with their father's name and their last name is *Smith*.

It is easy to see that for every database instance (either set- or bag-valued) the view V_1 (treated as query) contains the query Q, i.e. $Q \sqsubseteq_s V_1$ (and $Q \sqsubseteq_{bs} V_1$, $Q \sqsubseteq_b V_1$). Moreover, notice that Q is equivalent under set, bag, or bag-set semantics to the view V_2 (treated as query). Hence, there are three ways to answer the query Q using the above views. Firstly, we may pose Q directly on $\mathcal{D}$. Secondly, we may use the materialized view V_1; i.e. selecting the tuples from $V_1(\mathcal{D})$ such that the first name of each employee is the same with his father's name, we get the answer $Q(\mathcal{D})$. Finally, we may use the second view in order to get the desirable answer ($Q(\mathcal{D})$ is equal to $V_2(\mathcal{D})$). Notice that using the last two cases we avoid part of query computation; hence we speed up the query processing. □

Comparing the answers of two queries posed on an arbitrary database, the bag containment describes a tighter comparison than the same way containment under bag-set semantics; which describes a tighter comparison than the same way containment under set semantics. More specifically, see the following remark that is implied by the definition of the query containment.

Remark 4. Let two queries Q_1 and Q_2 over a database schema $\mathcal{S}$. Then the following hold:

- If $Q_2 \sqsubseteq_b Q_1$ then $Q_2 \sqsubseteq_{bs} Q_1$, and
- If $Q_2 \sqsubseteq_{bs} Q_1$ then $Q_2 \sqsubseteq_s Q_1$.

However, even if the two queries are CQs, the opposite direction of both cases of Remark 4 does not hold. The following example describes a case in which the set containment does not imply bag-set containment.

Example 10. Consider the queries Q_1 and Q_2 described by the following definitions:

$Q_1 : q(Age)$:- $student(Id, Age)$
$Q_2 : q(Age)$:- $student(Id, Age), emp(Id, Jobtitle)$

The first query results the ages of the students and the second one results the ages of the students that work in a job stored in the relation *emp*. It is easy to see that for every set-valued database instance $\mathcal{D}$ the answer of Q_2 when it is posed on $\mathcal{D}$, will be a subset of the answer $Q_1(\mathcal{D})$, i.e. $Q_2 \sqsubseteq_s Q_1$; because the age of a student that has a job will be also resulted by the first query. However, $Q_2 \not\sqsubseteq_{bs} Q_1$, since a student may have multiple jobs. □

The following example shows that bag-set containment does not imply bag containment.

Example 11. Consider the CQs Q_1, Q_2 with the following definitions, over the schema $\mathcal{S}$ that contains the binary relation *link*.

$Q_1 : q(X, Y)$:- $link(X, Y)$
$Q_2 : q(X, Y)$:- $link(X, Y), link(Y, Y)$

The relation $link(X, Y)$ stores pairs of Web-sites (X, Y) for which there is a link from X to Y. Moreover, the first query results the pair of sites that there is a link from the first site to the second site; and the second query results the pair of sites that there is a link from the first site to the second site and the second site has also a self-link.

It is easy to see that if the database instance is set-valued (i.e. we store in the relation *link* only whether or not there is a link from a Web-site to another), then we have that $Q_2 \sqsubseteq_{bs} Q_1$. However, each Web-site may link to itself multiple times (or there may be multiple links from a Web-site to another); which implies a bag-valued database instance. Hence, using the bag-valued database instance $\mathcal{D} = \{(link(a, b); 5), (link(b, b); 2)\}$, we see that $Q_2 \not\sqsubseteq_b Q_1$. □

2.3.1 Deciding conjunctive query containment

The decision of whether or not a query is contained in another query constitutes a problem that is extensively investigated for various classes of queries. In this section we focus on significant tools used for solving the conjunctive query containment. More specifically, we define the containment mapping from a CQ to another CQ and give the definition of the canonical database of a CQ. Moreover, we describe how these techniques are used for deciding query containment under set semantics and also give necessary and sufficient conditions for deciding query containment under bag, and bag-set semantics. The problem of conjunctive query containment under bag and bag-set semantics is further investigated in Chapter 3.

Containment mapping: Supposing two CQs over the same database schema the containment mapping describes a function from the set of variables of one CQ to the set of variables and constants of another CQ. The formal definition is given as follows.

Definition 4. Let two CQs Q_1 and Q_2 over the same schema $\mathcal{S}$. We say that a mapping $\mu : body(Q_1) \to body(Q_2)$ is a **containment mapping** from Q_1 to Q_2, denoted as $\mu : Q_1 \to Q_2$, if $\mu(head(Q_1)) = head(Q_2)$.

Example 12. Let the CQs Q_1 and Q_2 over a database schema $\mathcal{S}$ that contains the binary relation *edge*. The definitions of Q_1 and Q_2 are the following:

$$Q_1 : q(X) \text{ :- } edge(X,Y), edge(Y,Z), edge(Y,W)$$
$$Q_2 : q(A) \text{ :- } edge(A,B), edge(B,B), edge(B,C)$$

A graphical representation of the above queries is illustrated in Figure 2.3.

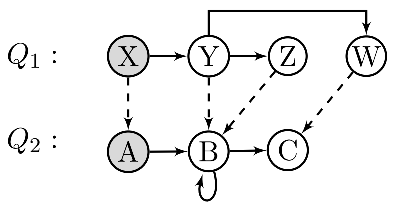


Figure 2.3: Graphical Representation of Q_1 and Q_2

Consider the mapping $\mu : body(Q_1) \to body(Q_2)$ such that $\mu = \{X/A, Y/B, Z/B, W/C\}$ (μ is represented in Figure 2.3 with dashed lines). Notice that μ is a containment mapping from Q_1 to Q_2. Moreover, considering an arbitrary database instance $\mathcal{D}$ of $\mathcal{S}$ and a valuation v from $\mathcal{D}$ on Q_2, it is easy to verify that the substitution $v' = \{X/v(\mu(X)), Y/v(\mu(Y)), Z/v(\mu(Z)), W/v(\mu(W))\}$ is a valuation from $\mathcal{D}$ on Q_1 such that $v(head(Q_2)) = v'(head(Q_1))$.

In addition, notice that there are totally four containment mappings from Q_1 to Q_2. However, there is no containment mapping from Q_2 to Q_1, since there is no way the second subgoal of Q_2 to map on a subgoal of Q_1. □

As the above example shows, the existence of a containment mapping from a CQ Q_1 to a CQ Q_2 guarantees that for every database instance $\mathcal{D}$ and for every valuation v from $\mathcal{D}$ on Q_2 there exists at least one valuation v' from $\mathcal{D}$ on Q_1 such that $v(head(Q_2)) = v'(head(Q_1))$. Consequently, the existence of a containment mapping constitutes a necessary and sufficient condition for conjunctive query containment under set semantics. The following theorem proves this assumption [CM77].

Theorem 1. *Let two CQs Q_1 and Q_2 over a database schema $\mathcal{S}$. The query Q_2 is contained in Q_1 under set semantics **if and only if** there is containment mapping from Q_1 to Q_2.*

Example 13. Continuing the Example 12, since there is a containment mapping from Q_1 to Q_2 then $Q_2 \sqsubseteq_s Q_1$. On the other hand, as there is no containment mapping from Q_2 to Q_1, we have that $Q_1 \not\sqsubseteq_s Q_2$. In particular, considering the database instance $\mathcal{D} = \{(1, 2)\}$ it is easy to see that $Q_2(\mathcal{D})$ is empty, instead of $Q_1(\mathcal{D})$ which contains the tuple $q(1)$; hence $Q_1(\mathcal{D}) \not\sqsubseteq_s Q_2(\mathcal{D})$. □

Instead of the case of set conjunctive query containment, the conjunctive query containment under bag and bag-set semantics is not decided via the existence of a containment mapping. Example 10 illustrates such a case (there is a containment mapping from Q_1 to Q_2, but $Q_2 \not\sqsubseteq_{bs} Q_1$). Moreover, notice that there is not containment mapping from Q_1 to Q_2 using which the variable *Jobtitle* of the second query is mapped.

Similarly, for every two CQs Q_1 and Q_2 such that although there is a containment mapping from Q_1 to Q_2, there is a variable Y of Q_2 that is not mapped using a containment mapping from Q_1, we can easily find a database instance that violates bag-set containment requirements. More specifically, we can easily see this by assigning multiple constants to Y and the same constant to any other variable of Q_2. The database instance constructed by the facts of Q_2's body causes a tuple in the answer of Q_2 whose multiplicity is greater than one; rather, the answer of Q_1 in which the same tuple appears once. The following proposition illustrates this necessary condition for bag-set CQ containment by showing that every variable of the candidate contained CQ have to be mapped using a containment mapping from the candidate containing CQ [CV93].

Proposition 1. *Let Q_1 and Q_2 be CQs over a database schema $\mathcal{S}$. If $Q_2 \sqsubseteq_{bs} Q_1$, then for every variable X in Q_2, there is a containment mapping μ from Q_1 to Q_2 such that $X \in \mu(Q_1)$.*

On the other hand, a property that guarantees bag-set CQ containment is that there is a containment mapping using which every variable of the candidate contained query is mapped. Formally, supposing two CQs Q_1, Q_2 over a database schema $\mathcal{S}$, we say that there is a containment mapping μ from Q_1 *variables-onto* Q_2 if for every variable X of Q_2 we have that X is in $\mu(Q_1)$. Proposition 2 proves the above sufficient condition for bag-set CQ containment [CV93].

Proposition 2. *Let Q_1 and Q_2 be CQs over a database schema $\mathcal{S}$. If there is a containment mapping from Q_1 variables-onto Q_2, then $Q_2 \sqsubseteq_{bs} Q_1$.*

Example 14. Continuing the Example 11, notice that the containment mapping $\mu = \{X/X, Y/Y\}$ from Q_1 to Q_2 is variables-onto; hence Proposition 2 implies that $Q_2 \sqsubseteq_{bs} Q_1$. □

The variables-onto containment mapping, however, is not also a necessary condition for bag-set CQ containment, since there are cases in which there is no variables-onto containment mapping and the CQ containment under bag-set semantics holds. The following example represent such a case.

Example 15. Consider the following CQs over the schema $\mathcal{S}$ that contains the binary relation r:

$$Q_1 : q(X,Y) \text{ :- } r(X,X'), r(Z,U), r(Z,W), r(Y,Y')$$
$$Q_2 : q(X,Y) \text{ :- } r(X,X'), r(X,U), r(Y,W), r(Y,Y')$$

Notice that there are two containment mappings from Q_1 to Q_2; in each of them the first subgoal of Q_1 maps on the first subgoal of Q_2 and the fourth subgoal of Q_1 maps on the fourth subgoal of Q_2. The second and the third subgoal of Q_1 maps together either on the second subgoal of Q_2 or on the third subgoal of Q_2. Hence, all subgoals of Q_2 are mapped using a containment mapping from Q_1, but there is no variables-onto containment mapping from Q_1 to Q_2. However, it is easy to verify that $Q_2 \sqsubseteq_{bs} Q_1$ holds (more details about why this containment holds are presented in Chapter 3). □

On the other hand, Remark 4 shows that the bag-set containment for two CQs constitutes a necessary condition for also deciding bag containment. Therefore, nor in the case of bag semantics the existence of a containment mapping suffice in order to decide bag CQ containment. However, for the case of bag CQ containment, a similar condition to that described by Proposition 1 holds. More specifically, if there is a subgoal of the candidate contained CQ that is not mapped using a containment mapping from the candidate containing CQ then the bag CQ containment does not hold. This necessary condition for bag CQ containment is proved by the following proposition [CV93]. The Proposition 3 also proves that the bag CQ containment implies a correspondence between the number of subgoals with same relation name appearing in two queries. The Example 16 illustrates two such cases.

Proposition 3. *Let Q_1 and Q_2 be CQs over a database schema $\mathcal{S}$. If $Q_2 \sqsubseteq_b Q_1$, then both the following hold:*

1. *for every subgoal g in Q_2, there is a containment mapping μ from Q_1 to Q_2 such that $g \in \mu(Q_1)$; and*
2. *for each relation name r, the number of subgoals with relation r in Q_1 is greater than or equal to the number of subgoals with relation r in Q_2.*

Example 16. Continuing the Example 11 it is easy to see that there is a unique containment mapping from Q_1 to Q_2 (which is also variables-onto), but the subgoal $link(Y, Y)$ of Q_2 is not mapped using any containment mapping from Q_1. Hence, the case 1 of Proposition 3 implies that $Q_2 \not\sqsubseteq_b Q_1$.

Now, consider the CQ Q_3 over the schema $\mathcal{S}$ with definition:

$$Q_3 : q(X, Y) \text{ :- } link(X, Y), link(Y, Y), link(Y, Z)$$

Notice that Q_3 contains Q_2 under bag semantics, since Q_3 results a tuple (x, y) for each link starts from y. However, as Q_3 has three subgoals with relation name $link$, instead of Q_2 that has two, the case 2 of Proposition 3 implies that $Q_3 \not\sqsubseteq_b Q_2$. □

For the case of bag semantics, the existence of a containment mapping from the candidate containing CQ to the candidate contained CQ such that every subgoal of the candidate contained CQ is mapped by a distinct subgoal of Q_1, guarantees bag CQ containment. We denote such a mapping as a subgoals-onto containment mapping; i.e. supposing two CQs Q_1 and Q_2 over a database schema $\mathcal{S}$, a containment mapping μ from Q_1 to Q_2 is *subgoals-onto* if *body*(Q_2) is a subexpression of *body*$(\mu(Q_1))$ (without eliminating duplicate subgoals). The following proposition formally shows this assumption [CV93].

Proposition 4. *Let Q_1 and Q_2 be CQs over a database schema $\mathcal{S}$. If there is a containment mapping from Q_1 subgoals-onto Q_2, then $Q_2 \sqsubseteq_b Q_1$.*

Example 17. Let the simple database schema $\mathcal{S}$ which models the Web graph and is also described in Example 11. Also, consider the CQs Q and Q' over $\mathcal{S}$ with the following definitions.

$$Q : q(A) \text{ :- } link(A, B), link(A, C)$$
$$Q' : q(X) \text{ :- } link(X, Y)$$

Notice that the containment mapping $\mu = \{A/X, B/Y, C/Y\}$ from Q to Q' is subgoals-onto; hence Proposition 4 implies that $Q' \sqsubseteq_b Q$. However, while there are two containment mappings from Q' to Q, none of them is subgoals-onto. □

The subgoals-onto containment mapping, however, is not a sufficient condition for bag CQ containment, since there are cases in which there is no subgoals-onto containment mapping and the CQ containment under bag semantics holds. The following example depicts such a case.

Example 18. Consider the CQs introduced in the Example 15. Notice that there is no subgoals-onto containment mapping from Q_1 to Q_2. However, it is easy to verify that Q_2 is bag contained in Q_1. □

Summarizing, the existence of a containment mapping is a necessary and sufficient condition for set CQ containment. However, it constitutes only necessary condition for bag-set and bag CQ containment. In [CM77], it is proved that the complexity of finding a containment mapping is NP-complete. For the case of bag-set and bag CQ containment, strengthening the definition of containment mapping two sufficient conditions are implied for CQ containment under these semantics. More specifically, the existence of a variables-onto containment mapping implies bag-set CQ containment and the existence of a subgoals-onto containment mapping implies bag CQ containment. In [CV93], it is proved that the complexity of finding either a variables-onto or a subgoals-onto containment mapping is also NP-complete.

Canonical Databases: A *canonical database* $\mathcal{D}$ of a CQ Q with respect to a set of constants C is a database instance obtained as follows: we turn each subgoal of Q into a fact by substituting each variable in the body of Q by a distinct constant from C, and treat the resulting subgoals, after eliminating duplicate facts, as the only tuples in $\mathcal{D}$. We refer to the valuation using which a canonical database of Q is constructed as *canonical*. The following proposition [AHV95, Ull89] shows how we can use the concept of canonical database to decide CQ containment under set semantics.

Proposition 5. *Let two CQs Q_1 and Q_2 over the same schema $\mathcal{S}$. Considering the canonical database $\mathcal{D}_C$ of Q_2, we have that $Q_2(\mathcal{D}_C) \subseteq_s Q_1(\mathcal{D}_C)$* ***if and only if*** *$Q_2 \sqsubseteq_s Q_1$.*

Example 19. Consider the queries Q_1 and Q_2 introduced in the Example 10. Constructing the canonical database $\mathcal{D}_C$ of Q_2 we use the lowercase letters a, i and j to denote constants. We then consider the canonical valuation v over Q_2 such that $v = \{Age/a,\ Id/i,\ Jobtitle/j\}$. Applying v on Q_2 the subgoals appearing in the body of $v(Q_2)$ constitute the $\mathcal{D}_C$; i.e. $\mathcal{D}_C = \{student(i,a), emp(i,j)\}$. The answer of Q_2 on $\mathcal{D}_C$ is $Q_2(\mathcal{D}_C) = \{q(a)\}$ which is equal to the answer $Q_1(\mathcal{D}_C)$. Hence, Proposition 5 implies that $Q_2 \sqsubseteq_s Q_1$. □

In Example 19 we noticed how the canonical database is used to decide set CQ containment. However, as we noticed in Example 10, Q_2 is not contained in Q_1 neither under bag-set nor under bag semantics. Therefore, we cannot use the technique of canonical database to decide either bag or bag-set CQ containment.

2.3.2 Conjunctive query equivalence

In this section we focus on deciding CQ equivalence under set, bag and bag-set semantics. Testing CQ equivalence under set semantics is straightforward. We check the set CQ containment for both directions and if the two-way containment holds then the CQs are set equivalent. In the case of bag and bag-set semantics we cannot decide the CQ equivalence in the same manner, since we don't even know how to check bag and bag-set containment in the class of CQs. However, in [CV93], it is proved that checking equivalence under both bag and bag-set semantics is much more easier.

The key concept used to decide CQ equivalence either under bag or under bag-set semantics is the isomorphic containment mapping. We say that a subgoals-onto containment mapping from a CQ Q_1 to a CQ Q_2 is *isomorphic* if it is also a renaming substitution. The following observation is implied by Case 2 of Proposition 3 and it is essential for deciding bag equivalence.

Remark 5. Considering two CQs Q_1, Q_2 over the same schema, if $Q_1 \equiv_b Q_2$ then for each relation name g the queries Q_1 and Q_2 have the same number of g-subgoals.

The Remark 5 is used to prove the following proposition; which shows that the existence of an isomorphic containment mapping between two CQs is a necessary and sufficient condition for deciding bag equivalence [CV93].

Proposition 6. *Let two CQs Q_1 and Q_2 over the same schema $\mathcal{S}$. There is an isomorphic containment mapping from Q_1 to Q_2* ***if and only if*** *$Q_1 \equiv_b Q_2$.*

Example 20. As we noticed in the Example 17, there is not any isomorphic containment mapping from Q and Q'; hence the Proposition 6 implies that $Q \not\equiv_b Q'$. Consider, now, the following CQ over the same schema $\mathcal{S}$.

$$Q'' : q(X) \text{ :- } link(X,Y), link(X,Z).$$

The containment mapping $\mu = \{A/X, B/Y, C/Z\}$ from Q to Q'' is isomorphic; thus, in this case Proposition 6 implies that $Q'' \equiv_b Q$. □

For the case of bag-set semantics deciding bag-set equivalence is almost the same, and it is based on the following important observation implied by the definition of variables-onto containment mapping.

Corollary 2. *Considering the CQs Q and Q^+ such that Q^+ is obtained by adding duplicate subgoals to Q, we have that $Q \equiv_{bs} Q^+$.*

The above corollary can be showed by easily proving that there are two variables-onto containment mappings one from Q to Q^+ and one from Q^+ to Q. We refer to the CQ that is obtained by eliminating duplicate subgoals from a CQ Q as the *canonical representation* of Q, denoted by Q^m. The following proposition proves that deciding the bag-set equivalence of two CQs Q_1 and Q_2 is equivalent to deciding whether the queries Q_1^m and Q_2^m are bag equivalent or not [CV93].

Proposition 7. *Let two CQs Q_1 and Q_2 over the same schema $\mathcal{S}$. There is an isomorphic containment mapping from Q_1^m to Q_2^m* ***if and only if*** *$Q_1 \equiv_{bs} Q_2$.*

Example 21. Consider two CQs P_1 and P_2 over the schema $\mathcal{S}$ introduced in the Example 11. The definitions of P_1 and P_2 are the following.

$$P_1 : p(X) \text{ :- } link(X,Y), link(Y,Y), link(Y,Y).$$
$$P_2 : p(A) \text{ :- } link(A,B), link(B,B), link(A,B).$$

Notice that there is no isomorphic containment mapping $\mu = \{A/X, B/Y\}$ from P_1 to P_2. Next, we produce the P_1^m and P_2^m by eliminating duplicate subgoals from P_1 and P_2, respectively. Hence, the definitions of P_1^m and P_2^m are the following.

$$P_1 : p(X) \text{ :- } link(X, Y), link(Y, Y).$$
$$P_2 : p(A) \text{ :- } link(A, B), link(B, B).$$

It is easy to see that the mapping μ is an isomorphic containment mapping from P_1^m to P_2^m. Thus, Proposition 7 implies that $P_2 \equiv_{bs} P_1$. □

The Propositions 6 and 7 shows that the bag and bag-set CQ equivalence problems are equivalent to the graph isomorphism problem [CV93]. Hence, the complexity of deciding either bag or bag-set CQ containment is in NP [GJ79].

2.3.3 Further related work on query containment and equivalence problems

Over the years, the problem of query containment under set semantics is investigated in depth by many researchers. Chandra and Merlin [CM77] proved that CQ query containment under set semantics is NP-complete. Later, many researchers investigated the problem for several super-classes of CQs [SY80, Klu88, vdM97].

Under bag and bag-set semantics, however, the problem was not studied so much. Chaudhuri and Vardi [CV93] showed that under both bag and bag-set semantics the CQ query containment problem is $\Pi_2^p - hard$. Ioannidis and Ramakrishnan [IR95] showed that the problem under bag semantics for CQs without self-joins is decidable and the containment can be checked by whether or not a subgoals-onto containment mapping exists. Moreover, they proved that the query containment problem is undecidable for union of CQs. In [JKV06], it is proved that the problem of containment of CQs with inequalities, under bag semantics, is undecidable. Besides, it is shown that the query-containment problem for CQs with inequalities, under bag semantics, is polynomial-time equivalent to the same problem under bag-set semantics.

The problem of equivalence of CQs under bag and bag-set semantics is also investigated in [CV93] (see Section 2.3.2). The set query equivalence problem is simply reduced on checking two way containment.

For proper aggregate queries (i.e., the queries require the computation of at least one aggregate function and necessarily outputs the result of this function), Cohen et al. [Coh05, CNS03] show how it is possible to reduce containment of a wide class of proper aggregate queries to equivalence. Using this result they also prove that the containment of aggregate queries is decidable if one of the following happens: (1) both queries are count, sum or max-queries, (2) both queries are conjunctive avg or cntd-queries without constants. Moreover, in this case, bag semantics are considered for the evaluation of the queries.

2.4 Answering queries using views

In this section we focus on describing the basic concepts and algorithms of rewriting queries using views; which constitutes the most representative technique of answering queries using views [Hal01]. The problem of answering queries using views can be stated as follows: given a query on a database schema and a set of view definitions (also called *viewset*) over the same schema, we want to answer the query using only the answers to the views.

The technique we use to answer queries using a given viewset (the queries and the view definitions are defined over the same schema) is based on transforming each of the given queries to a new query that is posed on the schema defined by views. Such a transformation is achieved by using the definitions of views and may guarantee that for every database instance the answers of the new queries are either part of or the same to the answers resulted by the initial queries. For a given query Q and a set of view definitions $\mathcal{V}$ we say that a query R is a *rewriting* of Q using $\mathcal{V}$ if it is posed only on view relations in $\mathcal{V}$ and its answer is related to the answer of Q. Using the concept of rewriting we formally define the contained and the equivalent rewriting for each semantics of a query using a viewset as follows [Hal01, ACGP07].

Definition 5. Let a query Q and a viewset $\mathcal{V}$ over a database schema $\mathcal{S}$. We say that a rewriting R of Q using $\mathcal{V}$ is a **set contained rewriting** (**bag contained rewriting**, **bag-set contained rewriting**, respectively) of Q using $\mathcal{V}$ if for every set-valued (bag-valued, set-valued, respectively) instance $\mathcal{D}$ of $\mathcal{S}$ we have that $Q(\mathcal{D}) \subseteq_s R(\mathcal{V}(\mathcal{D}))$ ($Q(\mathcal{D}) \subseteq_b R(\mathcal{V}(\mathcal{D}))$, $Q(\mathcal{D}) \subseteq_b R(\mathcal{V}(\mathcal{D}))$, respectively). Similarly, we say that R is a **set equivalent rewriting** (**bag equivalent rewriting**, **bag-set equivalent rewriting**, respectively) of Q using $\mathcal{V}$ if $Q(\mathcal{D}) =_s R(\mathcal{V}(\mathcal{D}))$ ($Q(\mathcal{D}) =_b R(\mathcal{V}(\mathcal{D}))$, $Q(\mathcal{D}) =_b R(\mathcal{V}(\mathcal{D}))$, respectively).

We denote by $\mathcal{V}(\mathcal{D})$ the database instance obtained by evaluating the definitions of the views in $\mathcal{V}$ over $\mathcal{D}$. In addition, in the case of bag-set rewriting we consider that $\mathcal{V}(\mathcal{D})$ is the bag-valued database obtained by evaluating under bag-set semantics the views in $\mathcal{V}$ over the set-valued database $\mathcal{D}$; and we consider bag semantics for the evaluation of the rewriting over the database $\mathcal{V}(\mathcal{D})$. In the following we focus on the equivalent rewriting of a given query using a given set of views considering that the languages of the query, the rewritings and the view definitions is the CQs.

The rewriting problem under each semantics is formally defined as follows. Given a viewset $\mathcal{V}$ and a CQ Q over a database schema $\mathcal{S}$, we want to find a CQ which is a set (resp. bag, bag-set) equivalent rewriting R of Q using $\mathcal{V}$.

Example 22. Continuing the Example 9 we consider the query R over the viewset $\mathcal{V} = \{V_1, V_2\}$ with the following definition.

$$R : q(X, Y, Y) \text{ :- } v_2(X, Y, Y)$$

It is easy to see that for every (either bag-valued or set-valued) database instance $\mathcal{D}$ of $\mathcal{S}$ we have that the answer $Q(\mathcal{D})$ is the same with the answer $R(\mathcal{V}(\mathcal{D}))$ (i.e. $Q(\mathcal{D}) =_b R(\mathcal{V}(\mathcal{D}))$ and $Q(\mathcal{D}) =_s R(\mathcal{V}(\mathcal{D}))$, respectively). Hence, R is a set, bag and bag-set equivalent rewriting of Q using $\mathcal{V}$. $\square$

Throughout this thesis we consider the closed-world assumption [AD98, LRO96, Hal01], in which views are materialized from base relations, rather than views describing sources in terms of abstract predicates, as is common when the open-world assumption is used [LRO96, AD98, LRO96, Hal01].

Supposing a CQ Q and a viewset $\mathcal{V}$ over a database schema $\mathcal{S}$, for every CQ R in terms of $\mathcal{V}$ we define a CQ over $\mathcal{S}$ which is used on checking whether or not R is either a contained or an equivalent rewriting of Q. In the following, we refer as *view-subgoal* to each subgoal of R that is referred on a view included in $\mathcal{V}$. In addition, we refer as a *definition* of a view-subgoal to the definition of the corresponding view in $\mathcal{V}$.

The *view-expansion* of a view-subgoal v of R is the CQ obtained by applying a substitution h over the definition V of v such that the atoms $head(h(V))$ and v are identical, and the non-distinguished variables of $h(V)$ are fresh variables (i.e. they are not used either by R or by any other view-expansion of a view-subgoal of R). We denote as $exp(R, v)$ the view-expansion of the view-subgoal v of R. Notice that if there is at least one non-distinguished variable in the definition of a view-subgoal then there are infinite view-expansions of a certain view-subgoal; considering that there are infinite ways to substitute a non-distinguished variable with a fresh variable. However, all the view-expansions of a certain view-subgoal are related each other by applying a renaming substitution over the non-distinguished variables. Therefore, supposing a predefined substitution over the definition of each view-subgoal v we refer to the unique view-expansion of v.

We, now, formally define the expansion of a CQ R in terms of $\mathcal{V}$.

Definition 6. Let a viewset $\mathcal{V}$ over a database schema $\mathcal{S}$ and a CQ R over $\mathcal{V}$. The **expansion** of R, denoted as R^{exp}, is a CQ obtained by replacing every view-subgoal of R with the body of its view-expansion.

The following proposition shows that given a viewset $\mathcal{V}$ and a CQ Q over a schema $\mathcal{S}$, deciding whether or not the expansion of a CQ R over $\mathcal{V}$ is set equivalent (bag equivalent, bag-set equivalent, respectively) to Q is necessary and sufficient condition for proving that R is a set equivalent rewriting (bag equivalent rewriting, bag-set equivalent rewriting, respectively) of Q [Hal01, ACGP07]. A similar condition holds for the case of contained rewriting.

Proposition 8. *Let a CQ Q and a viewset $\mathcal{V}$ over a database schema $\mathcal{S}$. Then the following hold.*

- *For every CQ R over $\mathcal{V}$ we have that R is a set contained rewriting (bag contained rewriting, bag-set contained rewriting, respectively) of Q* ***if and only if*** *$R^{exp} \sqsubseteq_s Q$ ($R^{exp} \sqsubseteq_b Q$, $R^{exp} \sqsubseteq_{bs} Q$, respectively).*
- *For every CQ R over $\mathcal{V}$ we have that R is a set equivalent rewriting (bag equivalent rewriting, bag-set equivalent rewriting, respectively) of Q* ***if and only if*** *$R^{exp} \equiv_s Q$ ($R^{exp} \equiv_b Q$, $R^{exp} \equiv_{bs} Q$, respectively).*

Example 23. Let the CQ Q and the views V_1, V_2 (suppose that $\mathcal{V} = \{V_1, V_2\}$) over a database schema $\mathcal{S}$. The definitions of Q, V_1, V_2 are the following.

$$Q : q(X) \text{ :- } p(X,Y), p(Y,Z), p(Z,W), p(W,X)$$
$$V_1 : v_1(X,Y) \text{ :- } p(X,Z), p(Z,Y)$$
$$V_2 : v_2(X,Y) \text{ :- } p(X,Z), p(Z,Y), p(Y,W)$$

Now, suppose the CQs R_1 and R_2 over $\mathcal{V}$ with the following definitions.

$$R_1 : q(A) \text{ :- } v_1(A,C), v_1(C,A)$$
$$R_2 : q(A) \text{ :- } v_1(A,C), v_2(C,A)$$

The definitions of each view-subgoal of R_1, R_2 with relation name v_1 is the definition of V_1. Similarly, the definition of the second view-subgoal of R_2 (named by v_2) is the definition of V_2. Moreover, using the substitutions $\theta_1 = \{X/A, Y/C, Z/F_1\}$, and $\theta_2 = \{X/C, Y/A, Z/F_2\}$ over the definition of V_1 we get the view-expansions of the first and second subgoal, respectively, of R_1; which are given by the following rules.

$$exp(R_1, v_1(A,C)) : v_1(A,C) \text{ :- } p(A,F_1), p(F_1,C)$$
$$exp(R_1, v_1(C,A)) : v_1(C,A) \text{ :- } p(C,F_2), p(F_2,A)$$

Similarly, the view-expansions of the view-subgoals of R_2 are the following.

$$exp(R_2, v_1(A,C)) : v_1(A,C) \text{ :- } p(A,F_3), p(F_3,C)$$
$$exp(R_2, v_2(C,A)) : v_2(C,A) \text{ :- } p(C,F_4), p(F_4,A), p(A,F_5)$$

Constructing the expansions of R_1^{exp} and R_2^{exp} from the view-expansions of their view-subgoals, we have the following rules.

$$R_1^{exp} : q(A) \text{ :- } p(A,F_1), p(F_1,C), p(C,F_2), p(F_2,A)$$
$$R_2^{exp} : q(A) \text{ :- } p(A,F_3), p(F_3,C), p(C,F_4), p(F_4,A), p(A,F_5)$$

Considering the mapping $\mu_1 = \{A/X,\ F_1/Y,\ C/Z,\ F_2/W\}$ it is easy to verify that μ_1 is an isomorphic containment mapping from R_1^{exp} to Q; hence Proposition 6 implies that $Q \equiv_b R_1^{exp}$ (consequently $Q \equiv_{bs} R_1^{exp}$ and $Q \equiv_s R_1^{exp}$). Thus, Proposition 8 implies that R_1 is a bag (also bag-set and set) equivalent rewriting of Q using $\mathcal{V}$.

Moreover, it is easy to see that there is no isomorphic containment mapping from R_2^{exp} to Q; hence R_2^{exp} is neither a bag nor a bag-set equivalent rewriting of Q. However, using the mappings $\mu_{21} = \{A/X,\ F_3/Y,\ C/Z,\ F_4/W,\ F_5/Y\}$ and $\mu_{22} = \{X/A,\ Y/F_3,\ Z/C,\ W/F_4\}$ from R_2^{exp} to Q and from Q to R_2^{exp}, respectively, it is easy to verify that μ_{21} and μ_{22} are containment mappings. Hence, Proposition 1 implies that $R_2^{exp} \equiv_s Q$; thus, Proposition 8 implies that R_2 is a set equivalent rewriting of Q using $\mathcal{V}$. $\square$

2.4.1 Useful views for rewriting a CQ

The common theme across all of the work on answering queries using views is that they all have to deal with the fundamental question of when a view is usable to answer a query. In this paragraph we focus on the minimum requirements that a conjunctive view meets in order to be used for constructing a useful (either a contained or a equivalent) rewriting of a CQ.

The Proposition 8, together with the containment and equivalence hierarchy over semantics (bag containment/equivalence implies bag-set containment/equivalence and bag-set containment/equivalence implies set containment/equivalence), give an important condition of when a view can be useful for a CQ (i.e. it may be used in either an equivalent or a contained rewriting of a CQ). Therefore, it is easy to see that the body of the view definition must be such an expression that the existence of a containment mapping from the query to the expansion of the rewriting is ensured (least requirement for the existence of a set contained rewriting).

Focusing on the views that can be used on constructing an equivalent rewriting the decision of whether a view is useful or not is easier than searching on useful views for a contained rewriting. The key notice is that there must also be a containment mapping from the expansion of the equivalent rewriting to the query. Hence, a view definition whose body does not map on a subexpression of the query's body cannot be used by any equivalent rewriting of the query. The following corollary is implied by the definition of containment mapping and the Proposition 8.

Corollary 3. *Let a CQ Q and a viewset $\mathcal{V}$ over the same database schema. For every set equivalent rewriting R of Q and for each view V in $\mathcal{V}$ that is* *used by R we have that there is a mapping from the body(V) to a subexpression of body(Q).*

Considering that S is the subexpression of *body*(Q) which is mapped by the body of V, we say that V *covers* S and each subgoal g of S (under set semantics).

In the case of bag semantics, as an isomorphic containment mapping is required in order to decide equivalence, a stronger sufficient condition of that described in the previous corollary holds. The following lemma shows that the existence of a substitution suffice (instead of the existence of a mapping) in order to decide whether a view is useful or not for equivalently rewriting a CQ under bag semantics [ACGP07].

Lemma 1. *Let a CQ Q and a viewset $\mathcal{V}$ over the same schema. For every bag equivalent rewriting R of Q using $\mathcal{V}$, we have that the body of the definition of each view V used R is a generalization of a subexpression S of Q.*

In this case, we similarly say that V *covers* S (under bag semantics). The following example illustrates how we can decide whether a view is useful or not for equivalently rewriting of CQ.

Example 24. Consider the query Q and the views V_1, V_2 defined on the Example 23. Also consider the views V_3, V_4 defined over the same schema $\mathcal{S}$ with the following definitions.

$$V_3 : v_3(A) \text{ :- } p(A,B), p(B,C)$$
$$V_4 : v_4(A,B,C) \text{ :- } p(A,B), p(B,B), p(B,C)$$

As we noticed in the Example 23 the views V_1, V_2 can be used for equivalently rewriting Q. In addition we observe that there are mappings from their bodies to subexpressions of the *body*(Q). Moreover, as the view V_1 can be used for constructing a bag equivalent rewriting, then Lemma 1 implies that the body of V_1 is a generalization of a subexpression of Q.

Focusing, now, on the view V_3, notice that using the substitution $\theta = \{A/X, B/Y, C/Z\}$ we show that the *body*($\theta(V_3)$) is a generalization of a subexpression of *body*(Q). Hence, using Lemma 1, one may think that V_3 may be useful for constructing a bag equivalent rewriting of Q. However, it is easy to verify that there is no bag equivalent rewriting which can be constructed by V_3. On the other hand, a set equivalent rewriting of Q that uses V_3 can be constructed. The following CQ constitutes such a rewriting.

$$R_3 : q(A) \text{ :- } v_1(A,C), v_1(C,A), v_3(A)$$

Concerning the view V_4 it is easy to verify that there is neither substitution nor mapping from *body*(V_4) to a subexpression of Q; hence Corollary 3 and Lemma 1 imply that there is no (bag, bag-set, or set) equivalent rewriting of Q which uses V_4. □

Summarizing the previous discussion searching on the existence of an equivalent rewriting of Q using $\mathcal{V}$ we focus on the set of views with definitions whose body map on a subexpression of the Q's body. Finally, for the existence of a bag equivalent rewriting we only focus on view definitions whose body is a generalization of Q's body.

2.4.2 On constructing equivalent rewritings

The construction, however, of an equivalent rewriting of CQ, under each semantics, also requires an appropriate combination of the views in a way such that the expansion of the produced rewriting is equivalent to the CQ. In this paragraph, we describe the requirements that a viewset meets in order to be used for equivalently rewriting a CQ.

Example 25. Let a database that stores information for car dealers. The schema $\mathcal{S}$ of the database contains the binary relations *car*, *loc* and the relation *part* of arity three. A tuple $car(m, d)$ means that the dealer d sells cars of make m. A tuple $loc(d, c)$ means that the dealer d has a branch in the city c; and a tuple $part(s, m, c)$ means that the store s in city c sells parts for cars of make m.

Consider that we submit the CQ Q that asks for cities and stores that sell parts for car makes in the *anderson* branch in this city. The definition of Q is given as follows.

$Q : q(S, C)$:- $car(M, anderson), loc(anderson, C), part(S, M, C)$

As we notice, the answer of Q is produced by applying selections on the relations *car* and *loc* such that the dealer is *anderson*, and then joining the results with the relation *part* such that the location in *loc* and *part* is the same and the make of *car* and *part* is the same. The answer is finally obtained by applying a projection on the stores and the cities that the stores are located.

In addition, suppose that we have the viewset $\mathcal{V}$ that contains the materialized views V_1, V_2, V_3, V_4, V_5 and V_6 with the following definitions.

$V_1 : v_1(S)$:- $car(M, anderson), loc(anderson, C), part(S, M, C)$
$V_2 : v_2(M, D, C, S)$:- $car(M, D), loc(D, C), part(S, M, C)$
$V_3 : v_3(M, D, C)$:- $car(M, D), loc(D, C)$
$V_4 : v_4(S, M, C)$:- $part(S, M, C)$
$V_5 : v_5(D, C)$:- $car(M, D), loc(D, C)$
$V_6 : v_2(S, C)$:- $part(S, M, C)$

Notice that the body of each of the above view definitions is generalization of a subexpression of Q's body.

Searching for equivalent rewritings of Q using $\mathcal{V}$ we see that the body of the first view is identical to that of query. However, the distinguished variables of the view V_1 is a subset of the variables needed in order to get the answer of the query (i.e. V_1 does not result the information about the cities that each store is located). Hence, we cannot equivalently rewrite Q using V_1 under each semantics.

The view V_2 applies the same join operators on the base relations with that the query applies. The answer of this view is a superset of the information needed for answering Q. Hence applying a selection on the answer of V_2 such that the dealer is *anderson*, and then projecting on the stores and their cities, for every database instance of $\mathcal{S}$ we get a result equal (under each semantics) to the answer of Q. More specifically, the definition of the set (also bag and bag-set) equivalent rewriting of Q using only the view V_2 is the following.

$$R_2 : q(S, C) \text{ :- } v_2(M, anderson, C, S)$$

Here, notice that we can use V_1, together with V_2 to construct a set equivalent rewriting of Q. More specifically, the definition of this rewriting is the following.

$$R_{12} : q(S, C) \text{ :- } v_2(M, anderson, C, S), v_1(S)$$

It is easy to verify there is a containment mapping from Q to R_{12}^{exp} and there is also a containment mapping from R_{12}^{exp} to Q.

The view V_3 answers the makes that a dealer sells, together with the cities in which the dealer has a branch. Moreover, the answer of V_4 is a copy of the relation *part*. Hence, selecting the tuples of V_3 such that the dealer is *anderson*, and then joining the result with the answer of V_4 such that each make appearing in V_3 is the same to that appearing in V_4 and the cities appearing in answers of both V_3 and V_4 are equal, we can get the answer of Q (we finally project to the stores and cities). Thus the corresponding set equivalent rewriting of Q using the views V_3 and V_4 is that with the following definition.

$$R_{34} : q(S, C) \text{ :- } v_3(M, anderson, C), v_4(S, M, C)$$

Focusing, now, on constructing a set (or bag or bag-set) equivalent rewriting of Q using V_5 and V_6, we observe that whereas the bodies of V_5 and V_6 are the same with V_3 and V_4, respectively, the information resulted by V_5 and V_6 is less than the information resulted by V_3 and V_4 (the makes of a certain dealer is not resulted in the answers of both views). More specifically, notice that the variable M is not a distinguished variable of V_5 and V_6. Hence, it is easy to see that we cannot answer Q by posing a CQ only in the answers of V_5 and V_6 (we cannot join their answers such that the makes are equal). Thus, there is no set (or bag, or bag-set) equivalent rewriting of Q using only V_5 and V_6. However, we easily see that we can construct a set equivalent rewriting of Q using V_5 and V_6 together with the views V_3 and V_4. The definition of such a rewriting (there is not only one) is the following.

$R_{3456} : q(S, C)$:- $v_3(M, anderson, C), v_4(S, M, C), v_5(D, C), v_6(S, C)$

Finally, since under set semantics the existence of containment mappings in both directions (i.e., from the expansion of the rewriting to Q and from Q to the expansion of the rewriting) suffice in order to have a set equivalent rewriting, we can construct set equivalent rewritings of Q using all views of $\mathcal{V}$. The definition of such a rewriting is given as follows.

$$\begin{aligned} R_{\mathcal{V}} : q(S, C) \text{ :- } & v_1(S), v_2(M, anderson, C, S), \\ & v_3(M, anderson, C), v_4(S, M, C), \\ & v_5(D, C), v_6(S, C) \end{aligned}$$

□

In the above example, notice that each view-subgoal in an equivalent rewriting is given by substituting the variables of definition of the view-subgoal using the corresponding mapping from its body to a subexpression of Q's body. This observations lead us to describe the way we can decide the existence of an equivalent rewriting, and it is formally captured by the concept of view tuple [ALU07].

Definition 7. Given a CQ Q, we obtain a canonical database D_Q of Q. Let $\mathcal{V}$ be a viewset. For each tuple in $\mathcal{V}(D_Q)$, we restore each introduced constant back to the original variable of Q, and the result of this replacement is called a **view tuple** of the query given the views. Moreover, we denote as $T(Q, \mathcal{V})$ the set of view tuples of Q and $\mathcal{V}$.

In [ALU07], it is proved that if there is a set equivalent rewriting of Q using $\mathcal{V}$, then there is a set equivalent rewriting whose head is identical to the head of Q and its subgoals are view tuples of Q and $\mathcal{V}$. The following proposition formally describes this result.

Proposition 9. *Let a CQ Q and a viewset $\mathcal{V}$. If there is a set equivalent rewriting R of Q using $\mathcal{V}$, then there is a set equivalent rewriting R' of Q using $\mathcal{V}$ such that $body(Q) = body(R')$ and each subgoal of R' is a view tuple of Q and $\mathcal{V}$.*

As an example, consider the canonical database D_Q of Q illustrated in Example 25. D_Q contains the tuples $car(m, anderson)$, $loc(anderson, c)$,

$part(s, m, c)$, where s, m and c are constants. Computing the view tuples of V_1 and V_2 we get $V_1(D_Q) = \{v_1(s)\}$ and $V_2(D_Q) = \{v_2(m, anderson, c, s)\}$; from which the view tuples $v_1(S)$, $v_2(M, anderson, C, S)$ are obtained. Notice that the CQ R_{12} is a set equivalent rewriting which contains the above view tuples in its body and its head is identical to Q.

Focusing on bag semantics, we can decide the existence of a bag equivalent rewriting much easier. The key notice for finding such a rewriting is given by the Proposition 6. Hence, combining the results of Lemma 1 and Proposition 6 we easily conclude that in each bag equivalent rewriting the view-expansions of its subgoals form a partition of the query subgoals. This result is formally given as follows [ACGP07].

Lemma 2. *Let Q be a CQ and $\mathcal{V}$ be viewset. There is either a bag equivalent rewriting R of Q in terms of $\mathcal{V}$* ***if and only if*** *the view-expansions of the subgoals of R form a partition of the subgoals of Q.*

Therefore, considering a CQ Q and a viewset $\mathcal{V}$, we can find either a bag or a set equivalent rewriting (if there is any) using the Proposition 9 (which also holds for both bag and bag-set semantics) and the above lemma. In particular, we firstly compute the view tuples of Q and $\mathcal{V}$. Then, searching for a bag equivalent rewriting we get a combination of view tuples that form a partition of subgoals of Q. On the other hand, searching for a set equivalent rewriting we get a combination of view tuples for which there are containment mappings in both directions (from Q to the produced rewriting and vice versa). In each case, the head of the constructed rewriting is identical to that of Q.

Minimal rewritings

The existence of multiple equivalent rewritings of a given query Q and a given viewset (e.g., see Example 25) leads on finding a representative subset of them that satisfies the practical requirements. As we noticed in Example 25, there may be multiple combinations of views that give equivalent rewritings. Moreover, in the case of set semantics, the arbitrary duplication of view-subgoals of an equivalent rewriting implies an infinite number of equivalent rewritings (since duplicating subgoals of a set equivalent rewriting, new set equivalent rewritings are produced). Are some of them, however, more significant? The answer is given by the purpose of rewriting queries using views; which is that of answering a query as efficiently as we can.

Since, now, the number of joins in a query affects significantly the efficiency of the query evaluation, we focus on the equivalent rewritings with the minimum number of joins. Hence, we say that an equivalent rewriting is *locally-minimal* if we cannot remove any of its subgoals and still retain equivalence to the query. For example, the rewritings R_2, R_{34} in Example 25 are locally-minimal; the R_{12} and R_{3456}, however, are not (because removing the last subgoal of R_{12} and the last two subgoals of R_{3456}, we also get set equivalent rewritings). Notice, in this case, that R_2 applies less join operators than R_{34}; hence, applying R_2 on V_2, we answer Q more efficiently than applying R_{34} on the views V_3 and V_4. In order, now, to capture these cases, we define the *globally-minimal rewriting* to be the equivalent rewriting with the minimum number of subgoals. Therefore, in previous example the only globally-minimal rewriting is the R_2.

Focusing on this perspective we categorize the useful views in two types: the containment-target views, and the filtering views. A view V is a *containment - target* view for a CQ Q if there exists either a set or a bag, or a bag-set equivalent rewriting P of Q (P uses V), and there is a containment mapping from Q to the expansion P^{exp} of P, such that V provides the image of at least one subgoal of Q under the mapping. Intuitively, a containment-target view covers at least one query subgoal. Covering all query subgoals (w.r.t. the semantics we consider) is enough to produce an equivalent rewriting of the query. A view is a *filtering* view for a query if it is not a containment-target view.

As an example of filtering view, notice that the view V_1 described in Example 25 is a filtering view for Q. In order to verify this, notice the equivalent rewriting R_{12}. We can see that constructing the R_{12}^{exp}, there is not subgoal of view-expansion of the V_1-subgoal of R_{12} which is mapped by a subgoal of Q.

In [ACGP07], the authors prove that there is not any filtering view in a bag equivalent rewriting of a CQ; which is easily implied by the Lemma 2. In addition, we conclude that each equivalent rewriting of CQ in the case of bag semantics is also a locally- and globally-minimal rewriting. For bag-set semantics, now, the filtering views cannot also exist in equivalent rewritings of CQs. This follows from the Proposition 7. In particular, the expansion of each bag-set equivalent rewriting R of a CQ Q contains subgoals that are the same with the subgoals of Q up to a renaming substitution. Moreover, in the case of bag-set equivalent rewriting there may be views that provide only duplicate subgoals in its expansion. However, it is easy to verify that in each locally-minimal bag-set equivalent rewriting these views do not exist [ACGP07].

However, the filtering views in a set equivalent rewriting may significantly reduce the evaluation time of answering a CQ [CHS02]. An example of such case is described as follows.

Example 26. Consider a hypothetical shipping company that serves a number of cities, with fixed delivery schedules between pairs of cities. Suppose the company has a centralized database, with a relation $t(Source, D, Dest)$ that stores all pairs of cities $Source$ and $Dest$, such that there is a scheduled delivery from $Source$ to $Dest$ on day D of the week (a number between 1 and 7).

Suppose that agents of the company try to contract shipments to independent truck drivers, by attracting them with tours connecting two or more cities. The company predefines a number of tour types to offer to the truck drivers, and agents need to query the database and find out whether the tour requested by the driver exists starting at a given city. Every tour type starts and ends in the same city.

Consider the "four cities in five days with a break" tour which is given by the following query.
$Q : q(X_1) \text{ :- } t(X_1, 1, X_2), t(X_2, 1, X_3), t(X_3, 2, X_4), t(X_4, 4, X_3), t(X_3, 5, X_1)$

In addition, consider the two views V_1 and V_2 such that V_1 is a copy of the base relation t, and V_2 stores the set of cities on a cycle of length 5 (i.e., the cities that can be toured in 5 days). The definitions of these views are the following.

$$\begin{aligned}
&V_1 : v_1(X, D, Y) \text{ :- } t(X, D, Y)\\
&V_2 : v_2(X) \text{ :- } t(X, D_1, X_1), t(X_1, D_2, X_2),\\
&\qquad\qquad t(X_2, D_3, X_3), t(X_3, D_4, X_4), t(X_4, D_5, X)
\end{aligned}$$

Suppose, now, the following rewritings R_1 and R_2.

$$\begin{aligned}
&R_1 : q(X_1) \text{ :- } v_1(X_1, 1, X_2), v_1(X_2, 1, X_3),\\
&\qquad\qquad v_1(X_3, 2, X_4), v_1(X_4, 4, X_3), v_1(X_3, 5, X_1)\\
&R_2 : q(X_1) \text{ :- } v_2(X_1), v_1(X_1, 1, X_2), v_1(X_2, 1, X_3), v_2(X_3),\\
&\qquad\qquad v_1(X_3, 2, X_4), v_1(X_4, 4, X_3), v_1(X_3, 5, X_1)
\end{aligned}$$

It is easy to verify that both rewritings are set equivalent rewritings of Q using V_1 and V_2. Moreover, since the view V_2 results the cities that can appear in a "five days" tour (given by the CQ Q), R_2 is used to speed up the query Q (instead of the rewriting R_1). The view V_2, however, is a filtering view and R_1 is a globally-minimal rewriting of Q using V_1 and V_2. Hence, in the case of set semantics, filtering views can be used to speed up CQ evaluation and globally-minimal rewritings do not necessarily describe the most efficient way of answering a CQ using views. □

2.5 Selecting conjunctive views

In many data-management scenarios, (such as information integration, data warehousing, Web-site designs, and query optimization) the views are used on reformulating a database for various aspects. Such a reformulation is achieved by constructing a set of views, which are defined on an initial database, and treating them as a new database [CG00]. For example, in query optimization, generating and materializing a set of views, we can significantly speed-up query evaluation. In information integration systems, reformulating the databases of the sources we succeed data integration. In this point of view, the purposes of reformulation can be represented by a set of constraints over the views. In this section, we focus on a database reformulation, called view selection problem, which is based on finding a viewset such that a given set of queries can be efficiently rewritten.

Intuitively, for a given set of queries over a database schema, the view selection problem is to choose a set of views to materialize (where the views are defined over the same database schema), such that a set of constraints over the views and the rewritings of the queries using the views, is satisfied. This version of database reformulation is widely used for query optimization purpose. Moreover, the hardness of this problem is caused by its multicriteria nature (multiple constraints may be given) and depends on the number and the nature of the given constraints. Several constraints are proposed in the bibliography (see Section 2.5.2); which differ with respect to the application's requirements. For example, we may want to construct the viewset with the minimum maintenance cost, or the viewset of the minimum size. In this thesis, we focus on two constraints, one over the viewset, and one over the equivalent rewritings. More specifically, the two criteria are: (1) for a given set of views, the selection of the less-costly equivalent rewritings of the queries and (2) the choice of the appropriate set of views which does not violate a storage constraint. This bicriteria version of the problem is formally given as follows.

Given a set $\mathcal{Q}$ of queries (also called *query workload*), defined on a schema $\mathcal{S}$, and a database instance $\mathcal{D}$ of $\mathcal{S}$, we want to find and precompute offline a viewset $\mathcal{V}$ defined on $\mathcal{S}$, such that the views in $\mathcal{V}$ can be used to compute the answers to all queries in the workload $\mathcal{Q}$. More specifically, the *view selection problem*, is to find a set of views that when materialized, (a) would satisfy a set $\mathcal{L}$ of constraints on the size of the views, and (b) can be used to get equivalent rewritings of the queries in $\mathcal{Q}$ which minimize the evaluation cost of the queries. We refer to the tuple $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ as the *input of view selection problem*. The view selection problem is said to be *bag-oriented* (resp. *set-oriented*, or *bag-set-oriented*) if we consider bag semantics (resp. set semantics, or bag-set semantics).

In addition, we consider that the only constraint on materialized views is a storage limit L (i.e. $\mathcal{L} = \{L\}$), which is a bound on the size of the views (which represents the available disk space for storing the views). Our goal is to choose the viewsets which minimize the evaluation cost of the queries and whose size will not exceed the limit L. Notice that, if the storage limit is sufficiently large then we can materialize all query answers, which is an optimal viewset. The problem becomes interesting when the storage limit is less than that. In the following we measure the size of a relation R as the number of tuples in R, and is denoted as $size(R)$.

Definition 8. Let $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ be a view selection problem input. A viewset $\mathcal{V}$ is said to be **admissible** for $\mathcal{P}$ if

1. $\mathcal{V}$ gives equivalent rewritings of all the queries in $\mathcal{Q}$,
2. for every view $V \in \mathcal{V}$, there exists at least one equivalent rewriting of a query in $\mathcal{Q}$ that uses V, and
3. $\mathcal{V}$ satisfies the constraints $\mathcal{L}$.

The following definition formally defines the *solution* and *optimal solution* of view selection problem for a given input.

Definition 9. Let a view selection problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$.

- A **solution** of $\mathcal{P}$ is a tuple $(\mathcal{V}_{adm}, \mathcal{R})$, where $\mathcal{V}_{adm}$ is an admissible viewset for $\mathcal{P}$ and $\mathcal{R}$ is a set of equivalent rewritings of the queries in $\mathcal{Q}$ using $\mathcal{V}_{adm}$.
- An **optimal solution** for $\mathcal{P}$ is a solution which minimizes the cost of evaluating the queries in the workload among all solutions of $\mathcal{P}$. The viewset in an optimal solution is said to be an **optimal viewset**.

We say that an equivalent rewriting of a query in $\mathcal{Q}$ using an admissible viewset is a *candidate rewriting*; and if the viewset is optimal, we refer to the less-costly equivalent rewritings as *optimal rewritings*.

Throughout this thesis we consider that the language of view definitions of queries and rewritings is the conjunctive queries.

As we can easily notice, solving the view selection problem requires explicit knowledge of the definitions of views in the viewsets which give equivalent rewritings of CQs in the input workload, together with the definitions of the corresponding rewritings. The viewsets, however, that give equivalent rewritings of a given CQ may be infinite. The equivalent rewritings using a specific viewset may be infinite as well. Therefore, we focus on describing a bounded space of viewsets that guarantees the existence of at least one optimal solution (if there is any).

Measuring efficiency of equivalent rewritings: Optimal solutions relate to the estimation of the cost of evaluating a query. We thus demand from the optimal solutions to minimize a given cost-function that we employ. We assume that the views have been precomputed, hence we do not assume any cost of computing them. For CQs we define the *sum-of-joins* cost model [ACGP07, CHS02] which measures the cost of query evaluation as the sum of the costs of all the joins in the evaluation. More specifically, suppose a query Q and a database $\mathcal{D}$. We assume use of only *left-linear query plans*, where selections are pushed as far as they go and projection is the last operation. Thus, each plan is a permutation of the subgoals of the query, and the cost of this query plan on a given database instance $\mathcal{D}$ is defined inductively as follows. For $n = 1$, the cost of query plan $Q = R_1$ is the size of the relation R_1. For each $n \geq 2$, the cost of query plan $(\ldots((R_1 \bowtie R_2) \bowtie R_3) \bowtie \ldots \bowtie R_n)$ over n relations is the sum of the following four values:

1. the cost of query plan $(\ldots((R_1 \bowtie R_2) \bowtie R_3) \bowtie \ldots \bowtie R_{n-1})$
2. the size of relation $R_1 \bowtie \ldots \bowtie R_{n-1}$
3. the size of relation R_n and
4. the size of relation $R_1 \bowtie \ldots \bowtie R_n$

The cost of evaluating a query Q on a database $\mathcal{D}$, denoted as $C(Q, \mathcal{D})$, is the minimum cost over all Q's query plans when evaluated on $\mathcal{D}$. Moreover, the cost of a query workload, denoted as $C(\mathcal{Q}, \mathcal{D})$, is defined as the sum of the costs of all queries in the workload.

In the bibliography, multiple cost models are proposed [GHRU97, CHS02, ACGP07, ALU07]. Most of them considers that the evaluation cost is increasing with the size of intermediate relations. We refer to each model that satisfies this property as *monotonic* cost model. Throughout this thesis we consider that each cost model is monotonic (except if explicitly mentioned).

The sum-of-joins cost model has two perspectives implied by the way the intermediate relations are constructed [ALU07]. In the first we consider that in each intermediate relation the attributes of the joined relations are retained [GMUW08]. In the second one, the nonrelevant attributes are dropped after the computation of each intermediate relation (these intermediate relations are called generalized supplementary relations) [BR91]. More specifically, after the first i subgoals are processed, the generalized supplementary relation is the intermediate relation with the nonrelevant attributes of i-th relation in the query plan dropped. Notice that computing a supplementary relation is essentially the same as doing projection push down in the execution of a physical plan for a query, which is a method supported by most optimizers.

On the other hand, a simpler cost model is also proposed in [ALU07]. This cost model (we refer to this as *num-of-subgoals* cost model) counts the number of subgoals and considers that the efficiency of the query is inversely increasing with the number of subgoals. In particular, in the case of equivalent rewritings, this cost model considers that the globally-minimal rewritings are the most efficient. However, as we noticed in Section 2.4.2, there are cases in which globally-minimal rewritings are not the most efficient ones (e.g., in the case of set semantics, filtering views may give optimal rewritings which are not globally-minimal). Throughout this thesis, we consider the sum-of-joins cost model (except if explicitly mentioned).

2.5.1 Determining the search space of solutions

Searching for optimal solutions we have to determine the search space of solutions for a given view selection problem input. In Section 2.4, we noticed that under set semantics both the admissible viewsets and the candidate rewritings of a CQ are infinite. In this case, considering either the sum-of-joins or the num-of-subgoals cost model, the optimal solutions for a given input may be infinite. An example of such a case is described as follows.

Example 27. Let the query workload $\mathcal{Q}$ and the set-oriented view selection problem input $\mathcal{P} = (\mathcal{S},\ \mathcal{Q},\ \mathcal{D},\ L)$. Moreover, consider the following queries that are contained in $\mathcal{Q}$.

$$Q_3 : q_3(X,Y) \text{ :- } p(X,Y), p(Y,W), p(W,X).$$
$$Q_4 : q_4(X,Y) \text{ :- } p(X,Y), p(Y,W), p(W,U), p(U,X).$$

Each query Q_i, with $i = 3, 4$, represents a cycle of length i. Also consider that $\mathcal{D} = \{p(a,a)\}$ and $L = 1$ tuple. From the required storage space we conclude that if there exists an optimal solution then each optimal viewset contains a single view. Moreover, notice that there is not any admissible viewset which is obtained by subexpression of a query's body (i.e., the bodies of the views are subexpressions of a query's body). Consider, now, the view V with the following definition.

$V : v(X_1, \ldots, X_{10})$:- $p(X_1, X_2), p(X_3, X_4), p(X_5, X_6), p(X_7, X_8), p(X_9, X_{10})$.

Considering the viewset $\mathcal{V} = \{V\}$ we can easily verify that $\mathcal{V}$ is an admissible viewset for $\mathcal{P}$. The candidate rewritings of the above two queries have the following definitions (the CQ R_i is the equivalent rewriting of Q_i using $\mathcal{V}$).

$R_3 : r_3(X, Y)$:- $v(X, Y, Y, W, W, X, X, Y, Y, W)$.
$R_4 : r_4(X, Y)$:- $v(X, Y, Y, W, W, U, U, X, X, Y)$.

Moreover, it is easy to see that considering either the sum-of-joins or the num-of-subgoals cost model these rewritings are optimal; hence, $\mathcal{V}$ is an optimal viewset. Similarly, for each $n \geq 4$, the viewset $\mathcal{V}_n$ that contains the view V_n with definition

$V : v(X_1, \ldots, X_n)$:- $p(X_1, X_2), p(X_3, X_4), \ldots, p(X_{n-1}, X_n)$,

is an optimal viewset for $\mathcal{P}$. □

Can we find, however, all optimal solutions by searching a bounded space of viewsets? The above example answers negatively the previous question, for the case of set-oriented view selection problem. Considering, however, queries without self-joins in the given workload the set-oriented view selection problem become tractable. In this case, if there exists any solution then there is an optimal solution such that none of the rewritings and views has self-joins. Moreover, considering that the number of subgoals of each given query does not exceed an integer n, the number of subgoals of each view definition in this optimal viewset is bounded from above by n [ACGP07]. In the set-oriented version, as we noticed in Section 2.4 the existence of filtering views in an optimal solution may also increase the complexity of the problem.

The complexity of the set-oriented view selection problem, when only queries without self-joins are included in the input workload and each rewriting does not include filtering views, is in NP [ACGP07]. This result follows from the fact that in this case we decide whether there exists any optimal solution, and find such a solution, by searching the generalizations of the bodies of the queries. More specifically, if there exists a solution we can find an optimal viewset in which each view definition (i.e., the body of its definition) is given by a generalization of a subexpression of the body of a query in the workload.

Focusing, now, on the bag-oriented version, the problem can be completely solved using the Lemmas 1 and 2. More specifically, constructing all views definitions whose body is a generalization of a subexpression of the body of a query in the workload we can find all admissible viewsets for a given input. This conclusion gives a sound and complete algorithm for solving the bag-oriented view selection problem. This algorithm, called CGALG [ACGP07], firsly constructs all view definitions whose body is a generalization of a subexpression of the body of a query in the workload. Then it constructs all the viewsets that contain such views and check for each of them whether or not it can fit into the available storage space (i.e., its size is at most equal to the storage limit). In the next step, the algorithm checks whether each of the viewsets satisfing the storage constraint give bag equivalent rewritings of all queries in the workload. The optimal rewritings are produced by comparing the efficiency of the bag equivalent rewritings during the previous test. In addition, notice that this algorithm is also sound for the set-oriented version where only queries without self-joins are included in the input workload and each rewriting does not include filtering views.

In the bag-set-oriented view selection problem, the authors in [ACGP07] also prove that if the CQs in the input workload contain self-joins then the bodies of the views in the optimal viewsets may have exponentially more subgoals than the definitions of the respective queries. The filtering views do not appear in a candidate rewriting (see also in Section 2.4). In the case that the input workload does not contain any query with self-joins, the problem can be completely solved. More specifically, we conclude from Proposition 7 that if there exists an admissible viewset for such a bag-set-oriented problem input, we can find an optimal solution such that the following hold [ACGP07]:

1. the body of each view definition appearing in the optimal viewset is a generalization of a subexpression of the body of a query in the workload,

2. the view definitions in the optimal viewset do not contain self-joins,

3. each optimal rewriting has at most the same number of subgoals with the query of the workload that it rewrites, and

4. rewriting any query in the workload does not require self-joins of view-subgoals.

From these results, it easily follows that the bag-set view selection problem is also in NP when none of the queries in the workload has self-joins. Moreover, we can easily notice that for such a view selection input, the CGALG algorithm is sound and complete (i.e., it completely solves the problem).

The bag- and bag-set-oriented versions of view selection problem are further investigated in Chapter 4.

2.5.2 Related work on selecting views

The problem of automatic selection of views to materialize has attracted the interest of many researchers. In [CG00], the space requirements for the view selection problem in the context of data warehouse design under set semantics, are considered. This paper, also investigates conditions under which the search space of optimal configurations can be reduced to the views that are subexpressions of the queries in the workload. In [TX04, XTZ06], the extraction of common subexpressions of the queries in the workload is studied. The authors in [TX04], study the problem of searching for a maximum common subexpression of a workload, while [XTZ06] proposes an algorithm for searching for maximum common subexpressions for a subclass of select-project-join SQL queries, using query graphs. Another approach for finding similar subexpressions is proposed in [ZLFL07] where workloads of select-project-join-groupby queries are considered. The authors propose a solution for the multi-query optimization problem which is incorporated in the Microsoft SQL Server. The algorithm has a lightweight mechanism (table-signatures) to detect common subexpressions and multiple sharing opportunities.

In [GM05], it is stated the view selection problem using AND-OR graphs to represent the query plans. Two types of constraints on materialized views are assumed, a storage limit and a maintenance-cost constraint. The candidate set of view configurations are given as input, hence the time of the construction of view configurations is not considered in the response time of the algorithms.

In [TS97], the view selection problem assuming a maintenance-cost constraint in the data warehouse environment and proposed an algorithm based on multi-query graphs, is studied. In [YCGL04], the authors examine greedy /heuristic algorithms for solving the view-selection problem assuming a maintenance-cost constraint and OLAP queries in multidimensional data warehouse environment. In [BPT97] the problem for multidimensional databases is studied and an algorithm that selects views by reducing significantly the solution space is proposed; considering only the relevant elements of the multidimensional lattice. The authors considered the standard SQL notion of group-by and aggregate functions in order to capture queries with aggregation. In earlier work, Rizzi and Saltarelli [EM03] presented a comparative evaluation that uses view materialization and indexing for a single GSPJ (Group-by-Select-Project-Join) query expressed on a star scheme for the data warehousing context.

The view selection problem, in the context of multidimensional data warehouses, also studied by several authors [KM99, HRU96, GHRU97]. In [KM99], it is described a system which was incorporated in Microsoft SQL Server and focuses on selection of both views and indexes. Earlier, the authors of [HRU96] propose algorithms for selecting views in the case of data cubes and study the complexity of the problem. In [GHRU97], the work of [HRU96] was further extended to include index selection.

A significant result that underlines the difference of the view selection problem in the case of queries with and without aggregation is presented in [AC05]. In this work, an algorithm for selecting views is proposed and complexity results are presented, using a theoretical approach to express GSPJ queries. The authors also showed that using materialized views to compute aggregate queries results greater benefits than for purely conjunctive queries; as a view with aggregation precomputes some of the grouping/aggregation on some of the query's subgoals.

In [CHS02], Chirkova et al. observed that the complexity of view selection problem under set semantics, and assuming conjunctive query workload, depends crucially on the quality of the estimates that a query optimizer has on the size of views. In [CHS02], it is also shown that an optimal choice of views may involve an exponential number of views in the size of the database schema. In the same context, in [ACGP07], Afrati et al. study the search space of candidate sets of views, under bag, set and bag-set semantics. Finally, the problem of selecting minimal-size-views to materialize has been studied theoretically in [CL03], where the problem has been proven decidable and an upper bound is given on this problem's complexity.

Chapter 3

Query containment under bag and bag-set semantics

Query containment was recognized fairly early as a fundamental problem in relational database query evaluation and optimization. In addition, this problem is closely related to many other database research topics including data integration and rewriting queries using views. The query containment problem has been extensively investigated for various types of database queries during the past two decades, but the focus is on set semantics (see Section 2.3). Conjunctive queries is one of the most interesting classes of SQL queries and the class that has been greatly investigated. For CQs under both bag and bag-set semantics the problem remains open for more than a decade after its definition by Chaudhuri and Vardi [CV93]. Most of the super-classes of CQs under these semantics, for which the problem have been investigated until now, give undecidability results (Section 2.3.3).

Focusing, now, in the class of CQs we can decide bag-set containment (resp. bag containment) by finding a variables-onto (resp. a subgoals-onto) containment mapping from one query to another (see in Section 2.3). However, we remind that the existence of a variables-onto (resp. a subgoals-onto) containment mapping is not a necessary condition for bag-set containment (resp. bag containment) (see also in Section 2.3). In this perspective, Chaudhuri and Vardi [CV93] proved that bag-set query containment implies a set of containment mappings such that every variable of the contained query is an image of the mapping (see also in Proposition 1). Similarly, bag query containment implies a set of containment mappings such that every subgoal of the contained query is obtained by applying one of these mappings to a subgoal of the containing query (see also in Proposition 3).

In this chapter, we investigate the conjunctive query containment problem under both bag and bag-set semantics through a careful analysis of special cases. In addition, we investigate the complexity of the problems for each such class and give necessary conditions for the conjunctive query containment problem. The main contributions of this chapter are summarized in Table 3.1.

Containing Query (Q_1)	**Contained Query (Q_2)**	**Complexity**	
		Bag-Set Semantics	**Bag Semantics**
CQ	CQ	$\Pi_2^p - hard$: [CV93]	
CQ	CQ without projections	NP Complete: Theorem 2	open
CQ without projections	CQ without projections	$O(n^2 log(n))$: Theorem 3+Remark 6	
CQ	CQ without self-joins	$O(nlog(n))$: Theorem 4	$O(nlog(n))$: [IR95], Theorem 5
Pathstar Query	Pathstar Query	$O(n^2 log(n))$: Theorem 7	$O(n^2 log(n))$: Theorem 8
Simple Gener.-Pathstar Query	Simple Pathstar Query	Linear: Theorem 9	Linear: Theorem 10
CQ	Enhanced Q_1	Linear: Theorem 13	open

Table 3.1: Complexity results for the CQ containment problem.

3.1 Contained query without projections

In this section we investigate the problem of deciding query containment under bag and bag-set semantics for a large subclass of CQs, namely the CQs that do not contain projections. This class contains the CQs whose body variables also appear in their heads. We prove that the results of evaluating such queries on any set-valued database are always sets; hence to decide bag-set containment of such queries it suffices to decide containment under set semantics. Concerning bag semantics, we prove that when both queries are without projections then to decide query containment it is sufficient to check for a subgoals-onto containment mapping.

Proposition 10. *Let Q be a CQ on a database schema $\mathcal{S}$ which is evaluated under bag-set semantics. Then the following are equivalent:*

1. *Each variable of the body of Q appears in its head.*
2. *For every set-valued database instance of $\mathcal{S}$ the following holds: the answer of Q is a set.*

Proof. Suppose that the query Q is of the form:

$$Q : q(\overline{X}) \text{ :- } g_1(\overline{X}_1), \ldots, g_n(\overline{X}_n)$$

(**From 2 to 1**) Suppose that for every set-valued database instance $\mathcal{D}$, $Q(\mathcal{D})$ is a set. We will prove by contradiction that $vars(\overline{X}_j) \subseteq vars(\overline{X})$ for every $j \in \{1, \ldots, n\}$. Suppose that there is a variable Y in a subgoal $g_j(\overline{X}_j)$ such that $Y \notin vars(\overline{X})$. Consider the database instance constructed as follows: Let $\mathcal{D}_c$ be a canonical set-valued database built from the body of Q, and let c_Y be the constant used to replace the variable Y in the body of Q in order to get $\mathcal{D}_c$. Let $\mathcal{D}_Y$ be the set of all atoms in $\mathcal{D}_c$ in which c_Y appears. Now let $\mathcal{D}'_c$ be the set of atoms obtained from $\mathcal{D}_Y$ by replacing all occurrences of c_Y by a new constant c'_Y which does not appear in $\mathcal{D}_c$. It is easy to see that $\mathcal{D}_c \cup \mathcal{D}'_c$ is a set-valued database instance as $\mathcal{D}_c$ is set-valued and the atoms in $\mathcal{D}'{}_c$ do not appear in $\mathcal{D}_c$. Besides $Q(\mathcal{D}_c \cup \mathcal{D}'_c)$ contains duplicates (as the use of the atoms in $\mathcal{D}'_c$ returns the same results as those obtained by using the corresponding atoms in $\mathcal{D}_Y$) which contradicts with our assumption.

(**From 1 to 2**) Suppose that $vars(\overline{X}_j) \subseteq vars(\overline{X})$ for every $j \in \{1, \ldots n\}$. We will prove by contradiction that, for every set-valued database instance $\mathcal{D}$, the answer of $Q(\mathcal{D})$ is a set. Since Q is safe, every head-variable of Q appears in its body. Thus $vars(head(Q)) = vars(body(Q))$. Suppose that, for a set-valued database instance $\mathcal{D}$, $Q(\mathcal{D})$ is not a set. Then there is a tuple t that appears in $Q(\mathcal{D})$ at least twice. But as $vars(head(Q)) = vars(body(Q))$, all occurrences of the tuple t correspond to the same instance of the query body. Thus, there should exist an instance of some body atom $g_i(\overline{X}_i)$ appearing in $\mathcal{D}$ at least twice. But this contradicts with the assumption that $\mathcal{D}$ is set-valued. □

Proposition 10 reveals the source of duplicate tuples in query evaluation under bag-set semantics, which is the existence of projection operators in the query. Theorem 2 immediately follows from Proposition 10. It shows that when there are no projections in the candidate contained query, we can decide containment under bag-set semantics by testing containment under set semantics (i.e. by searching for a containment mapping); which reveals, in this case, the complexity of the problem.

Theorem 2. *Let Q_1 and Q_2 be two CQs, such that every variable appearing in the body of Q_2 also appears in the head of Q_2. Then, the following holds: $Q_2 \sqsubseteq_s Q_1$ **if and only if** $Q_2 \sqsubseteq_{bs} Q_1$. The query containment problem , in this case, is NP-complete.*

By applying the Theorem 2 in the queries of the Example 11 we see that all variables in the body of Q_2 appear in its head and there is a containment mapping from Q_1 to Q_2 (hence $Q_2 \sqsubseteq_s Q_1$). Therefore $Q_2 \sqsubseteq_{bs} Q_1$.

Theorem 3. *Let Q_1 and Q_2 be CQs, such that all variables appearing in the body of Q_1 (resp. Q_2) also appear in the head of Q_1 (resp. Q_2). Then, $Q_2 \sqsubseteq_b Q_1$ **if and only if** there is a subgoals-onto containment mapping from Q_1 to Q_2. The complexity of testing containment is $O(n^2 \cdot log(n))$.*

Proof. **(If part)** From Proposition 4 we know that if there is a subgoals-onto containment mapping from Q_1 to Q_2 then $Q_2 \sqsubseteq_b Q_1$.

(Only-If part) (By contradiction) Suppose that $Q_2 \sqsubseteq_b Q_1$. Suppose also that there is no subgoals-onto containment mapping from Q_1 to Q_2. We will prove that this implies $Q_2 \not\sqsubseteq_b Q_1$ by finding a database instance $\mathcal{D}$ such that $Q_2(\mathcal{D}) \not\subseteq_b Q_1(\mathcal{D})$.

As $Q_2 \sqsubseteq_b Q_1$, we have that $Q_2 \sqsubseteq_{bs} Q_1$; hence Remark 4 implies that there is a containment mapping μ from Q_1 to Q_2. Moreover, since all variables of the queries also appear in their heads, by definition of containment mapping we have that μ is unique. In addition, Proposition 3 implies that each subgoal of query Q_2 is mapped by a subgoal of Q_1 using μ; i.e. every subgoal of Q_2 appears in $body(\mu(Q_1))$. Hence, supposing that μ is not subgoals-onto, we have that $body(Q_2)$ is not a subexpression of $body(\mu(Q_1))$. Thus, there is a subgoal g in Q_2 such that the number n_1 of subgoals of Q_1 that map g, using μ, is less than the number n_2 of duplicates of g in Q_2, i.e. $n_2 > n_1$.

Let Q'_2 be the CQ obtained by removing all duplicate subgoals from Q_2. Consider the canonical database $\mathcal{D}_c$ of Q'_2 and let the tuple t be the canonical instance of $\mu(g)$ in $\mathcal{D}_c$. Let $q(a_1, \ldots, a_n)$ a tuple in $Q_2(\mathcal{D}_c)$. Since $Q_2 \sqsubseteq_b Q_1$, then $q(a_1, \ldots, a_n)$ also belongs to $Q_1(\mathcal{D}_c)$. Since $\mathcal{D}_c$ is a set-valued database then Proposition 10 implies that $m_1 = m_2 = 1$, where m_i is the multiplicity of $q(a_1, \ldots, a_n)$ in $Q_i(\mathcal{D}_c)$ for $i = 1, 2$. Now, suppose that we add one more tuple t in $\mathcal{D}_c$, and let $\mathcal{D}$ be the new bag-valued database instance. It is easy to see that $m'_1 = 2^{n_1} < m'_2 = 2^{n_2}$, where m'_i is the multiplicity of $q(a_1, \ldots, a_n)$ in $Q_i(\mathcal{D})$ for $i = 1, 2$; which is a contradiction.

Complexity: The existence of a subgoals-onto containment mapping can be decided by the following algorithm: The unique mapping μ is found by mapping the i-th argument of the head of Q_1 to the i-th argument of the head of Q_2. We now rename the variables of Q_1 by applying the mapping μ on Q_1. Next we order the subgoals of $\mu(Q_1)$ and Q_2 lexicographically w.r.t. the relation names and the variables of each subgoal. Next we check if μ is a subgoals-onto mapping by traversing the subgoals of Q_1 and Q_2 in their lexicographic ordering and testing the following. For each sequence of duplicate subgoals in each query body it is retain only a single occurrence of the subgoal paired with an integer indicating the number of its occurrences in the query. In a second pass the queries are traversed and it is checked if both queries contain the same subgoals and the number of occurrences of each subgoal in Q_1 is greater than or equal to the number of the occurrences of the same subgoal in Q_2. If this is so then the test succeeds and then $Q_2 \sqsubseteq_b Q_1$; otherwise it fails and then $Q_2 \not\sqsubseteq_b Q_1$. As the ordering of the subgoals can be done in time $O(n^2 \cdot log(n))$ and each traversing is done in time $O(n)$, we see that the algorithm runs in time $O(n^2 \cdot log(n))$. □

Remark 6. The same algorithm used to prove the complexity results of Theorem 3 can also be used to prove bag-set query containment when both queries are CQs without projections. Hence, in this case, the bag-set query containment problem is in $O(n^2 \cdot log(n))$.

Example 28 shows that for bag semantics the requirement that both CQs are without projections is essential.

Example 28. Consider the following queries:

$$Q_1 : q(X,Y) \text{ :- } r(X,Y), r(Z,U), r(Z,W)$$
$$Q_2 : q(X,Y) \text{ :- } r(X,Y), r(X,X), r(Y,Y)$$

Here there is no subgoals-onto containment-mapping from Q_1 to Q_2. However, there are two containment mappings from Q_1 to Q_2 (the first subgoal of Q_1 maps the first subgoal of Q_2 and the other two subgoals of Q_1 map, together, either the second or the third subgoal of Q_2) such that all subgoals of Q_2 are mapped using a containment mapping from Q_1. However $Q_2 \sqsubseteq_b Q_1$. □

3.2 Contained Queries without self-joins

In this section we investigate the query containment problem under bag-set semantics when the candidate contained query does not contain self-joins. Our results are based on the following remark.

Remark 7. Let Q_1, Q_2 be CQs under bag-set semantics such that Q_2 does not contain self-joins. If there is a containment mapping μ from Q_1 to Q_2 then μ is unique.

The remark immediately follows from the fact that if there are two different containment mappings μ and μ' from Q_1 to Q_2 then there should be a subgoal g of Q_1 that can map to two different subgoals of Q_2. But this is impossible as Q_2 does not have self-joins.

Theorem 4. *Let Q_1 and Q_2 be two CQs such that Q_2 does not contain self-joins. Then, the following holds: $Q_2 \sqsubseteq_{bs} Q_1$* ***if and only if*** *there is a containment mapping from Q_1 variables-onto Q_2. Then we test containment in time $O(n \cdot log(n))$.*

Proof. The one direction, that is, if there is a variables-onto containment mapping from Q_1 to Q_2 then $Q_2 \sqsubseteq_{bs} Q_1$, immediately follows from Proposition 2.

For the other direction, suppose that $Q_2 \sqsubseteq_{bs} Q_1$. Suppose now that there is no variables-onto containment mapping from Q_1 to Q_2.

Case 1: There is no containment mapping at all from Q_1 to Q_2 as it contradicts with Proposition 1.

Case 2: There are containment mappings but none of them is variables-onto. In this case, Remark 7 implies that the containment mapping should be unique. Let μ be this mapping. As μ is not variables-onto, there is a variable X in Q_2, for which there is no variable X' of Q_1 such that $\mu(X') = X$. As μ is unique, this is impossible because it contradicts with Proposition 1.

Therefore, we conclude that if $Q_2 \sqsubseteq_{bs} Q_1$, then there is a variables-onto containment mapping from Q_1 to Q_2.

Complexity: The complexity of the containment test, is computed by the following algorithm: We sort the bodies of the queries w.r.t. relation names. This is done in $O(n \cdot log(n))$. Then we check whether or not there is a variables-onto containment mapping from Q_1 to Q_2 as follows: for each subgoal g of Q_1 we check whether or not the variables of g map the variables of the subgoal with the same relation name of Q_2, and check whether this mapping is a containment mapping; if not then $Q_2 \not\sqsubseteq_{bs} Q_1$. Otherwise, going through the subgoals of Q_2, we check if there is any variable of Q_2 which is not mapped on by Q_1. This can be done in $O(n)$. Hence, the complexity of the algorithm is $O(n \cdot log(n))$. □

The test proposed in Theorem 5 is due to [IR95]. Here, we observe that the algorithm used in Theorem 4 can also be used here. Hence the complexity is $O(n \cdot log(n))$.

Theorem 5. *Let Q_1 and Q_2 be two CQs such that Q_2 does not contain self-joins. Then, the following holds: $Q_2 \sqsubseteq_{b} Q_1$* ***if and only if*** *there is a containment mapping from Q_1 subgoals-onto Q_2. Then, we test containment in time $O(n \cdot log(n))$.*

3.3 Generalized-Pathstar Queries

In this section we introduce the notion of generalized-pathstar queries. Then we investigate the problem of checking containment under bag and bag-set semantics for specific subclasses of generalized-pathstar queries.

Definition 10. A query Q is called **generalized-pathstar query of arity** n, if it is of the following form:

$$\begin{aligned} Q : q(X_1, \ldots, X_n) \text{ :- } & S_1(X_1), \ldots, S_n(X_n), \\ & N_1(Y_1), \ldots, N_m(Y_m). \end{aligned}$$

with $n \geq 1$, and $m \geq 0$, where $X_1, \ldots, X_n$, $Y_1, \ldots, Y_m$ are distinct variables. Each S_i is called **distinguished pathstar** (or **d-star** for short) while each N_i is called **non-distinguished pathstar** (or **n-star** for short).

Each d-star (resp. n-star) consists of a sequence of labeled paths starting from the same distinguished variable X_i (resp. non-distinguished variable Y_i), called the **root of the pathstar**. A **labeled path** is an expression of the form $r_1(W_0, W_1)$, $r_2(W_1, W_2)$, $\ldots$, $r_k(W_{k-1}, W_k)$, with $k \geq 1$, where $r_1, r_2, \ldots, r_k$ are not necessarily distinct relation names, and $W_0, W_1, \ldots, W_k$ are distinct variables. If $m = 0$, then the query is called **pathstar query**. A (generalized-)pathstar query is called **simple** if every S_i and every N_i consists of paths of length 1 (i.e. each path is of the form $r(W_0, W_1)$). Finally, the number of subgoals in a pathstar S is called the **size** of S denoted as $size(S)$.

3.3.1 Pathstar queries

In this section we investigate the problem of deciding bag-set and bag query containment for pathstar queries containing paths of arbitrary length. In these cases, bag-set and bag containment can be tested in $O(n^2 \cdot log(n))$.

Theorem 6. *Let Q_1 and Q_2 be two pathstar queries of arity 1. Then, $Q_2 \sqsubseteq_{bs} Q_1$ **if and only if** there is a variables-onto containment mapping from Q_1 to Q_2. Then we test bag-set query containment in $O(n^2 \cdot log(n))$.*

Proof. **(If part)** It immediately follows from Proposition 2.

(Only if part) We will prove that if $Q_2 \sqsubseteq_{bs} Q_1$ then there is a variables-onto containment mapping from Q_1 to Q_2. For this it suffices to prove that: *"Each labeled path p of Q_2 which appears (up to renaming of its variables) n_2 times in Q_2, also appears n_1 times in Q_1, where $n_1 \geq n_2$"*. If this holds then we obtain a variables-onto mapping in which each path p of Q_2 is mapped on from a different path p of Q_1 covering in this way all variables of Q_2. Besides, all the other paths of Q_1 (not existing in Q_2) should map to prefixes of paths of Q_2 as otherwise there should be no containment mapping. But this contradicts with the assumption that $Q_2 \sqsubseteq_{bs} Q_1$.

We now prove the above statement by contradiction. For this, assuming that $n_1 < n_2$ for a particular path p, we find a database $\mathcal{D}$ for which $Q_2(\mathcal{D}) \not\sqsubseteq_b Q_1(\mathcal{D})$ which contradicts with the assumption that $Q_2 \sqsubseteq_{bs} Q_1$.

Let $\mathcal{D}$ be the database instance which is the canonical database of the query Q'_2 obtained from Q_2 by eliminating all but one occurrences of the path p. Suppose that the path p is of the form $r_1(Y, W_1)$, $r_2(W_1, W_2)$, $\ldots$, $r_k(W_{k-1}, W_k)$, with $k \geq 1$. Let $\mathcal{D}'$ be a database instance obtained from $\mathcal{D}$ by adding to the tuple $r_k(b, c_1)$, obtained as the canonical instance of $r_k(W_{k-1}, W_k)$, the tuples $r_k(b, c_2), \ldots, r_k(b, c_\ell)$ for some $\ell \geq 1$. Let $q(a)$ a tuple in $Q_2(\mathcal{D})$. Since $Q_2 \sqsubseteq_{bs} Q_1$, then $q(a)$ also belongs to $Q_1(\mathcal{D})$ and $m_1 \geq m_2$ where m_i is the multiplicity of $q(a)$ in $Q_i(\mathcal{D})$ for $i = 1, 2$. Moreover, suppose that $\lambda_i \geq 0$, where $i = 1, 2$, is the number of valuations from Q_i to $\mathcal{D}$ which map the head of Q_i to the tuple $q(a)$ and do not map any variable of Q_i to the constant c_1. It is easy to see that for the multiplicity m'_i of $q(a)$ in $Q_i(\mathcal{D}')$, for $i = 1, 2$, it holds $m'_i = (\ell)^{n_i} \cdot (m_i - \lambda_i) + \lambda_i$.

Let us now check if there is a value of ℓ for which $m'_2 > m'_1$, i.e. $(\ell)^{n_2} \cdot (m_2 - \lambda_2) + \lambda_2 > (\ell)^{n_1} \cdot (m_1 - \lambda_1) + \lambda_1$. Notice that $(\ell)^{n_2} \cdot (m_2 - \lambda_2) + \lambda_2 \geq (\ell)^{n_2} \cdot (m_2 - \lambda_2)$ and $(\ell)^{n_1} \cdot m_1 = (\ell)^{n_1} \cdot (m_1 - \lambda_1) + \lambda_1 \cdot (\ell)^{n_1} \geq (\ell)^{n_1} \cdot (m_1 - \lambda_1) + \lambda_1$. Hence, if $(\ell)^{n_2} \cdot (m_2 - \lambda_2) > (\ell)^{n_1} \cdot m_1$ or equivalently $(\ell)^{n_2 - n_1} > m_1/(m_2 - \lambda_2)$, then $m'_2 > m'_1$. Since it is easy to see that $m_2 > \lambda_2$ and we assumed that $n_2 < n_1$, we conclude that for a value of ℓ such that $\ell > \sqrt[n_2 - n_1]{m_1/(m_2 - \lambda_2)}$ it holds that $m'_2 > m'_1$. But this contradicts with the assumption that $Q_2 \sqsubseteq_{bs} Q_1$.

Complexity: Complexity is checked as follows: We count the occurrences of each labeled-path both in Q_1 and Q_2 as follows: We append the predicate names in each path and sort the resulting strings lexicographically and then we count the number of occurrences of each such string. Then we compare, properly, the corresponding numbers of occurrences. We can see that this procedure can be done in $O(n^2 \cdot log(n))$. □

Theorem 6 can be generalized for pathstar queries of arity n, with $n \geq 1$, as follows (its proof is constructed by extending the proof of Theorem 6):

Theorem 7. *Let Q_1 and Q_2 be pathstar queries of arity n, with $n \geq 1$. Then, $Q_2 \sqsubseteq_{bs} Q_1$* ***if and only if*** *there is a variables-onto containment mapping from Q_1 to Q_2. Then we test bag-set query containment in $O(n^2 \cdot log(n))$.*

The above results also hold for bag semantics. This is stated in Theorem 8 which is proved in a similar way.

Theorem 8. *Let Q_1 and Q_2 be pathstar queries of arity n, with $n \geq 1$. Then, $Q_2 \sqsubseteq_b Q_1$* ***if and only if*** *there is a subgoals-onto containment mapping from Q_1 to Q_2. Then we test bag query containment in $O(n^2 \cdot log(n))$.*

3.3.2 Simple generalized-pathstar queries

In this section we investigate the problem of deciding query containment under bag-set and bag semantics when the containing query belongs to the class of simple generalized-pathstar queries and the candidate contained query is a simple pathstar query. We prove that in this case we can test (bag-set or bag) containment in linear time.

Lemma 3. *Let Q_1 be a simple generalized-pathstar query and Q_2 be a simple pathstar query of the same arity n, with $n \geq 1$. Suppose that both queries are defined over a database schema consisting of a single binary relation r. Then $Q_2 \sqsubseteq_{bs} Q_1$* ***if and only if*** *for every subset $\mathcal{S}'$ of d-stars of Q_2 and the set $\mathcal{S}$ of the corresponding d-stars of Q_1 and the n-stars $N_1, \ldots, N_m$ of Q_1 the following inequality holds:*

$$\sum_{S' \in \mathcal{S}'} \text{size}(S') \leq \sum_{S \in \mathcal{S}} \text{size}(S) + \sum_{j=1}^{m} \text{size}(N_j) \tag{I}$$

Proof. The proof of the theorem is based on the observation that the multiplicity n of a specific tuple $t = q(a_1, \ldots, a_n)$ in $Q_1(\mathcal{D})$, where $\mathcal{D}$ is a database instance and Q_1 is a simple generalized-pathstar query of the form:

$$Q_1 : q(X_1, \ldots, X_n) \text{ :- } S_1(X_1), \ldots, S_n(X_n), N_1(Y_1), \ldots, N_m(Y_m).$$

is given by the formula:

$$n_1 = \prod_{i=1}^{n} \mathcal{M}(S_i(a_i)) \cdot \prod_{j=1}^{m} [\sum_{x \in \{a_1,\ldots,a_n\}} \mathcal{M}(N_j(x)) + \sum_{y \in (const(\mathcal{D}) - \{a_1,\ldots,a_n\})} \mathcal{M}(N_j(y))] \tag{II}$$

where $\mathcal{M}(S_i(a))$ (resp. $\mathcal{M}(N_i(a))$), we denote the number of distinct valuations over the d-star (resp. n-star) $S_i(X)$ (resp. $N_i(X)$) from the body of the query Q_1 to the database $\mathcal{D}$ which map X to the constant a.

In the case of a simple pathstar query of the form:

$$Q_2 : q(X_1, \ldots, X_n) \text{ :- } S'_1(X_1), \ldots, S'_n(X_n).$$

the multiplicity n_2 of t in $Q_2(\mathcal{D})$ is given by the formula:

$$n_2 = \prod_{i=1}^{n} \mathcal{M}(S'_i(a_i)). \tag{III}$$

The above formulas are based on the observation that in a simple generalized-pathstar query all non-distinguished variables, which are not roots of the pathstars, are disjoint.

(If part) Suppose that the inequality (I) holds. We will prove that $Q_2 \sqsubseteq_{bs} Q_1$ by showing that for every set-valued database instance $\mathcal{D}$, we have that $Q_2(\mathcal{D}) \subseteq_b Q_1(\mathcal{D})$; i.e. for every tuple $t = (a_1, \ldots a_n)$ in $Q_2(\mathcal{D})$, the multiplicity n_1 of t in $Q_1(\mathcal{D})$ is greater than or equal to the multiplicity n_2 of t in $Q_2(\mathcal{D})$.

Assume that $a'_1, \ldots a'_{n'}$ are the distinct constants appearing in t and let $\mathcal{J} = \{1, \ldots, n'\}$. Also assume that $\mathcal{D}$ contains τ_j distinct tuples of the form $r(a'_j, c)$, for every $j \in \mathcal{J}$. It is easy to verify that for every $j \in \mathcal{J}$, and for each d-star (resp. n-star) S of the queries, $\mathcal{M}(S(a'_j)) = \tau_j^{size(S)}$. For every $j \in \mathcal{J}$, let $\mathcal{S}_j$ (resp. $\mathcal{S}'_j$) be the set of d-stars of Q_1 (resp. the corresponding set of d-stars of Q_2), whose distinguished variables are valuated to the constant a'_j. Let also $\text{size}(\mathcal{S}_j) = \sum_{S \in \mathcal{S}_j} \text{size}(S)$. Finally, let $s_N = \sum_{j=1}^{m} \text{size}(N_j)$, and $\tau_\mu = max(\tau_1, \ldots, \tau_{n'})$, with $\mu \in \mathcal{J}$. It is now easy to see that formula (II) implies, the following inequality:

$$n_1 \geq \tau_\mu^{\text{size}(\mathcal{S}_\mu)} \cdot \Big(\prod_{i \in (\mathcal{J} - \{\mu\})} \tau_i^{\text{size}(\mathcal{S}_i)}\Big) \cdot \tau_\mu^{s_N} \tag{IV}$$

Similarly, formula (III) implies the equality:

$$n_2 = \tau_\mu^{\text{size}(\mathcal{S}'_\mu)} \cdot \prod_{i \in (\mathcal{J} - \{\mu\})} \tau_i^{\text{size}(\mathcal{S}'_i)} \tag{V}$$

As the inequality (I) holds for every set $\mathcal{S}'$ of d-stars of Q_2 and the set $\mathcal{S}$ of the corresponding d-stars of Q_1, we have that $s = \text{size}(\mathcal{S}_\mu) - \text{size}(\mathcal{S}'_\mu) + s_N \geq 0$. Combining (IV) and (V), we get the following inequality:

$$\frac{n_1}{n_2} \geq \tau_\mu^{s} \cdot \prod_{i \in (\mathcal{J} - \{\mu\})} \tau_i^{\text{size}(\mathcal{S}_i) - \text{size}(\mathcal{S}'_i)} \tag{VI}$$

Supposing that $\tau_\varepsilon = min(\tau_1, \ldots, \tau_{n'})$, with $\varepsilon \in \mathcal{J}$; and $\mathcal{I} = \{i | i \in (\mathcal{J} - \{\mu\})$ such that $\text{size}(\mathcal{S}_i) - \text{size}(\mathcal{S}'_i) < 0\}$, then from (VI) follows:

$$\frac{n_1}{n_2} \geq \tau_\mu^s \cdot (\prod_{i \in \mathcal{I}} \tau_\mu^{\text{size}(\mathcal{S}_i) - \text{size}(\mathcal{S}'_i)}) \cdot (\prod_{i \in (\mathcal{J} - (\mathcal{I} \cup \{\mu\}))} \tau_\varepsilon^{\text{size}(\mathcal{S}_i) - \text{size}(\mathcal{S}'_i)})$$

Using the inequality (I) we have that $s + \sum_{i \in \mathcal{I}} \text{size}(\mathcal{S}_i) - \text{size}(\mathcal{S}'_i) \geq 0$; hence $\frac{n_1}{n_2} \geq 1$. Thus $n_1 \geq n_2$.

(Only-If part) (Proof by contradiction). Suppose that $Q_2 \sqsubseteq_{bs} Q_1$. Assuming that the inequality (I) does not hold we will find a database instance $\mathcal{D}$ for which $Q_2(\mathcal{D}) \not\sqsubseteq_b Q_1(\mathcal{D})$.

Let $\mathcal{S}'$ be a set of d-stars of Q_2 such that $\sum_{S' \in \mathcal{S}'} \text{size}(S') > \sum_{S \in \mathcal{S}} \text{size}(S) + \sum_{j=1}^{m} \text{size}(N_j)$, where $\mathcal{S}$ is the set of the d-stars of Q_1 corresponding to $\mathcal{S}'$. Let $\mathcal{D} = \{r(a, c_1), \ldots, r(a, c_\lambda), r(b, b)\}$, where $\lambda \geq 1$. Assume, without lost of generality, that $\mathcal{S}'$ contains the first k d-stars of Q_2, with $n \geq k \geq 1$. Let now t' be the tuple $q(a, \ldots, a, b, \ldots, b)$ which has k occurrences of the constant a followed by $n - k$ occurrences of the constant b. Let n_1, n_2 be the multiplicities of t' in $Q_1(\mathcal{D})$, $Q_2(\mathcal{D})$ respectively. Applying formula (II) we get:

$$n_1 = (\prod_{i=1}^{k} \mathcal{M}(S_i(a))) \cdot (\prod_{i=k+1}^{n} \mathcal{M}(S_i(b))) \cdot \\ \cdot \prod_{j=1}^{m} (\mathcal{M}(N_j(a)) + \mathcal{M}(N_j(b)))$$

Assuming that s_i (resp. s'_i) is the size of the d-star S_i (resp. S'_i) , for $i = 1, \ldots, n$, and u_i is the size of the n-star N_i, for $i = 1, \ldots, m$, then

$$n_1 = (\prod_{i=1}^{k} \lambda^{s_i}) \cdot (\prod_{i=k+1}^{n} 1) \cdot \prod_{j=1}^{m} (\lambda^{u_j} + 1)$$

Hence, as $1 \leq \lambda^{u_i}$, for $i = 1, \ldots, m$, we conclude that $n_1 \leq \lambda^{s_1 + \cdots + s_k} \cdot 2^m \cdot \lambda^{u_1 + \cdots + u_m}$. By defining $s = \sum_{i=1}^{k} s_i$, and $u = \sum_{j=1}^{m} u_j$, the above inequality becomes $n_1 \leq 2^m \cdot \lambda^{s+u}$. In a similar way we get $n_2 = \lambda^{s'}$, where $s' = \sum_{i=1}^{k} s'_i$.

Let us now check if there is a value of λ for which $n_1 < n_2$, i.e. $2^m \cdot \lambda^{s+u} < \lambda^{s'}$; which can be written as $\lambda^{s'-s-u} > 2^m$. Since we assumed that $s' > s + u$, we conclude that for a value of λ such that $\lambda > \sqrt[s'-s-u]{2^m}$ we have $n_2 > n_1$; which is a contradiction. □

Theorem 9. *The bag-set query containment problem when the containing query Q_1 is a simple generalized-pathstar query and the contained query Q_2 is a simple pathstar query of the same arity n, with $n \geq 1$ and both queries are defined over a database schema consisting of a single binary relation r can be checked in linear time.*

Proof. The test introduced by the inequality (I) of Lemma 3 can be checked using the following algorithm: At first we calculate the differences $d_i = size(S_i) - size(S'_i)$, for $i = 1, \ldots, n$, of the corresponding d-stars of Q_1 and Q_2. Then we calculate the sum s of the negative values in $\{d_1, \ldots, d_n\}$. Finally, we compute the total size s_N of the n-stars in Q_1. Then if $|s| \geq s_N$ we conclude that $Q_2 \sqsubseteq_{bs} Q_1$; otherwise $Q_2 \not\sqsubseteq_{bs} Q_1$. It is easy to see that this algorithm runs in linear time with respect to the total number of subgoals in Q_1 and Q_2. □

The above results also hold for bag semantics as the following Theorem 10 states. The proof of this theorem is similar to the proof of Theorem 9 (and Lemma 3).

Theorem 10. *Let Q_1 be a simple generalized-pathstar query and Q_2 be a simple pathstar query of the same arity n, with $n \geq 1$. Suppose that both queries are defined over a database schema consisting of a single binary relation r. Then $Q_2 \sqsubseteq_b Q_1$* ***if and only if*** *for every subset $\mathcal{S}'$ of d-stars of Q_2 and the set $\mathcal{S}$ of the corresponding d-stars of Q_1 and the n-stars $N_1, \ldots, N_m$ of Q_1 the following inequality holds:*

$$\sum_{S' \in \mathcal{S}'} \text{size}(S') \leq \sum_{S \in \mathcal{S}} \text{size}(S) + \sum_{j=1}^{m} \text{size}(N_j)$$

Then we test bag query containment in linear time.

As an example we can apply the above results to the queries in Example 15. In particular, based on Lemma 3, we can easily verify that $Q_2 \sqsubseteq_{bs} Q_1$, while based on Theorem 10 we can see that $Q_2 \sqsubseteq_b Q_1$.

3.4 Other syntactic restrictions

In this section we further investigate the problem of query containment under bag-set semantics when the candidate contained query may contain self-joins. In particular, in Theorems 11 and 12 we present two necessary syntactic conditions for bag-set query containment. Theorem 13 shows that for a contained query which can be constructed by adding subgoals to the containing query, the bag-set query containment can be decided in linear time. Theorem 14 presents another case in which we can decide bag-set query containment. Finally, Proposition 11 presents a case in which we can decide non-containment under bag-set semantics.

Theorem 11. *Let Q_1 and Q_2 be CQs. Suppose that for a relation name r of Q_2, S_r is a non-empty set of non-distinguished variables of Q_2 such that each variable in S_r appears only in r-subgoals of Q_2 and for every two variables Y and Z in S_r there is no subgoal of Q_2 that contains both Y and Z. If $Q_2 \sqsubseteq_{bs} Q_1$ then the number of variables in S_r is less than or equal to the number of r-subgoals of Q_1.*

Proof. We will prove the theorem by contradiction. Let r a relation name of Q_2 and λ be the number of r-subgoals of Q_2 that contain a variable in S_r; and n_i be the number of r-subgoals of Q_i, where $i = 1, 2$. In addition, let $S_r = \{Y_1, \ldots, Y_k\}$, where $k = |S_r|$. Supposing that $k > n_1$, we will construct a database instance $\mathcal{D}$ for which there is a tuple in $Q_2(\mathcal{D})$ whose multiplicity is greater than the multiplicity of the same tuple in $Q_1(\mathcal{D})$.

It is easy to see that $\lambda \geq k$, because otherwise there should be two or more subgoals that contain a certain variable in S_r. Moreover, because of the assumptions of the theorem we have $n_2 \geq \lambda \geq k > n_1 \geq 1$.

Now consider a set-valued database instance $\mathcal{D}$ constructed from the body of Q_2 as follows: we consider $(k\ell + 1)$ distinct constants a, $b_{1,i}$, $b_{2,i}$, $\ldots$, $b_{k,i}$, where $i \in \{1, \ldots, \ell\}$. Then each variable $Z \in (vars(Q_2) - S_r)$ is replaced, in the body of Q_2, by the constant a. Now, we get all possible instances of the subgoals of Q_2 by replacing each $Y_j \in S_r$ by each one of the constants $b_{j,i}$, where $i \in \{1, \ldots, \ell\}$ and $j \in \{1, \ldots, k\}$. The instance $\mathcal{D}$ contains all ground atoms obtained by the above process. It is easy to see that, $\mathcal{D}$ contains at most $(\lambda\ell + 1)$ ground atoms (tuples) belonging to the relation r and one ground atom for each of the other relations in Q_2 (because the variables in S_r appear only in r-subgoals).

Because of Proposition 1, we know that for each variable X of Q_2 there is a containment mapping μ from Q_1 to Q_2 such that $X \in \mu(Q_1)$. We thus conclude that at least, every subgoal g_j, with $j \in \{1, \ldots \lambda\}$, of Q_2 that contains a variable in S_r, is mapped on from Q_1. Hence, the multiplicity m_1 of the tuple $t = q(\overline{a}) \in Q_1(\mathcal{D})$ is **at most** $(\lambda\ell+1)^{n_1}$ because there are at most $\prod_{i=1}^{n_1}(\lambda\ell+1) = (\lambda\ell+1)^{n_1}$ valuations over Q_1. Similarly, the multiplicity m_2 of the tuple $t = q(\overline{a}) \in Q_2(\mathcal{D})$ is **at least** $\prod_{i=1}^{\lambda}(\ell+1) = (\ell+1)^{\lambda}$.

In order to prove that $Q_2 \not\sqsubseteq_{bs} Q_1$ we have to prove that $m_2 > m_1$ i.e. $(\ell+1)^{\lambda} > (\lambda\ell+1)^{n_1}$; or that $(\ell+1)^{\lambda} > (\lambda(\ell+1))^{n_1} > (\lambda\ell+1)^{n_1}$. Consequently, by the last inequality, we have that $\ell > \sqrt[(\lambda-n_1)]{\lambda^{n_1}} - 1$. Moreover, since $\lambda > n_1 \geq 1$, we have that $\lambda \geq 2$. Hence $\ell > \sqrt[(\lambda-n_1)]{\lambda^{n_1}} - 1 > 1$. Thus, since the number ℓ is not fixed we choose $\ell = \lceil \sqrt[(\lambda-n_1)]{\lambda^{n_1}} \rceil$, where $\lceil x \rceil$ means the smallest integer which is greater than x. Consequently, there is a database instance $\mathcal{D}$ such that $Q_2(\mathcal{D}) \not\sqsubseteq_b Q_1(\mathcal{D})$. □

We now present another necessary syntactic condition for deciding bag-set query containment. The condition states that the containing query should contain at least as many variables as the contained query.

Theorem 12. *Let Q_1 and Q_2 be CQs. If $Q_2 \sqsubseteq_{bs} Q_1$ then the number of variables of Q_1 is greater than or equal to the number of variables of Q_2.*

Proof. We prove by contradiction that if $|vars(Q_2)| > |vars(Q_1)|$, then there is a database instance $\mathcal{D}$ such that the sum of multiplicities of the number of tuples of $Q_1(\mathcal{D})$ is less than the sum of multiplicities of the number of tuples of $Q_2(\mathcal{D})$; thus $Q_2 \not\sqsubseteq_{bs} Q_1$.

We construct $\mathcal{D}$ as follows: Let the set $A = \{a, b\}$, where a and b are distinct constants. For every relation name r in the bodies of Q_1 or Q_2, with arity k, we have that the relation instance $r(\mathcal{D})$ is:

$$r(\mathcal{D}) = \underbrace{A \times A \times \cdots \times A}_{k}.$$

Let $n_i = |vars(Q_i)|$, where $i = 1, 2$. Then, there are 2^{n_i} different valuations over Q_i, where $i = 1, 2$; as every variable X of Q_1 and Q_2 has two possible values (a and b). Therefore, $|Q_i(\mathcal{D})| = 2^{n_i}$, where $i = 1, 2$. As $n_2 > n_1$, we have that $|Q_2(\mathcal{D})| > |Q_1(\mathcal{D})|$. Thus, $Q_2(\mathcal{D}) \not\sqsubseteq_b Q_1(\mathcal{D})$. Consequently, $Q_2 \not\sqsubseteq_{bs} Q_1$. □

Theorems 11 and 12 can be used to decide non-containment for a wide subclass of CQs with self-joints. The following example depicts such situations.

Example 29. Consider the queries Q_1 and Q_2:

$$Q_1 : q(X) \text{ :- } r(X, Y), s(X, Z), s(X, W)$$
$$Q_2 : q(X) \text{ :- } r(X, Y), r(X, Z), s(X, X)$$

Observe that $|vars(Q_1)| > |vars(Q_2)|$. Thus, from Theorem 12, we conclude that $Q_1 \not\sqsubseteq_{bs} Q_2$. On the other hand, using Theorem 11 we conclude that $Q_2 \not\sqsubseteq_{bs} Q_1$ as the number of variables in $S_r = \{Y, Z\}$ is not less than or equal to the number of r-subgoals in Q_1. Notice that Theorem 12 cannot be used to prove that $Q_2 \not\sqsubseteq_{bs} Q_1$. □

Proposition 11. *Let Q_1 and Q_2 be CQs such that there is a variable $Y \in vars(Q_2)$ which does not appear in any subgoal of Q_2 whose relation name also appears in Q_1. Then $Q_2 \not\sqsubseteq_{bs} Q_1$*

Proof. As the variable Y does not appear in any subgoal of Q_2 whose relation name appears in the body of Q_1, we conclude that there is no containment mapping μ from Q_1 to Q_2 with the property that there is a variable Y' of Q_1 such that $\mu(Y') = Y$. Thus, Proposition 1 implies that $Q_2 \not\sqsubseteq_{bs} Q_1$. □

Let Q_1 and Q_2 be CQs. Then Q_2 is said to be Q_1-*enhanced* if it is obtained by adding a sequence of subgoals to Q_1. The following theorem shows that in this case the existence of a variables-onto containment mapping is a necessary condition for query containment.

Theorem 13. *Let Q_1 and Q_2 be CQs such that Q_2 is Q_1-enhanced. Then $Q_2 \sqsubseteq_{bs} Q_1$* ***if and only if*** *there is a variables-onto containment mapping from Q_1 to Q_2. In this case we can test bag-set query containment in linear time.*

Proof. **(If part)** Proposition 2 implies that if there is a variables-onto containment mapping from Q_1 to Q_2 then $Q_2 \sqsubseteq_{bs} Q_1$.

(Only-If part) We now prove the inverse by contradiction. Let Q_2 is obtained by adding a sequence R of subgoals to the body of Q_1. Suppose that $Q_2 \sqsubseteq_{bs} Q_1$ but there is no variables-onto containment mapping from Q_1 to Q_2. Hence, there is at least one variable in $vars(R)$ that does not appear in $S = Q_2 - R$, because S is identical to Q_1. In this case $|vars(Q_2)| > |vars(Q_1)|$. Then, Theorem 12 implies that $Q_2 \not\sqsubseteq_{bs} Q_1$.

Complexity: We can test bag-set containment by simply counting the number of variables in each query and testing if $|vars(Q_1)| > |vars(Q_2)|$. This can be done in linear time. □

Example 30. Consider the queries Q_1 and Q_2:

$$Q_1 : q(X, Y) \text{ :- } edge(X, Y)$$
$$Q_2 : q(X, Y) \text{ :- } edge(X, Y), edge(X, Z)$$

Notice that Q_2 is obtained by adding the atom $edge(X, Z)$ to Q_2. Notice also that there is no variables-onto containment mapping from Q_1 to Q_2; hence Theorem 13 implies that $Q_2 \not\sqsubseteq_{bs} Q_1$. □

Theorem 14 shows that by adding subgoals, which do not introduce new variables, to the body of a bag-set contained query we get a bag-set contained query.

Theorem 14. *Let Q_1 and Q_2 be two CQs, such that $Q_2 \sqsubseteq_{bs} Q_1$. Let Q_3 be a CQ obtained by adding the new subgoals $g_1(\overline{X}_1), \ldots, g_n(\overline{X}_n)$, with $n \geq 1$, to the body of Q_2, such that $vars(\overline{X}_i) \subseteq vars(Q_2)$ for $i = 1, \ldots, n$. Then $Q_3 \sqsubseteq_{bs} Q_1$.*

Proof. By construction of Q_3, there is a containment mapping from Q_2 variables-onto Q_3. Thus, from Proposition 2 we conclude that $Q_3 \sqsubseteq_{bs} Q_2$. As $Q_2 \sqsubseteq_{bs} Q_1$ we conclude that $Q_3 \sqsubseteq_{bs} Q_1$. □

3.5 Conclusions and Future Work

In this chapter we have studied the problem of query containment for CQs under both bag and bag-set semantics. In particular, we gave necessary and sufficient conditions for testing bag and bag-set containment for several major subclasses of CQs, and find the complexity for these cases. We also proved important properties that can be used to decide bag-set query containment, or guarantee non-containment. The decidability of the problem of testing containment of CQs under either bag or bag-set semantics remains open in the general case.

Moreover, an interesting problem to study is the bag-set or bag query containment problem in the presence of certain constraints on the relational schema [JK84]; e.g. the presence of key and foreign key constraints. We expect that the additional information given by the keys and foreign keys may imply interesting results about testing query containment under these semantics.

Another open problem arises by noticing that Proposition 10 proves a stronger claim than the claim needed to prove Theorem 2. In particular, it asserts additionally that for a CQ Q to return set result on every set-valued database D it is necessary to contain only distinguished variables. It is interesting to investigate constraints on database instances under which wider classes of CQs return sets instead of bags. This is important as in this case set containment implies bag-set containment hence the latter can be tested by finding a single containment mapping.

In addition, in this chapter we investigated the bag and bag-set containment problem for pathstar and simple generalized pathstar queries (i.e., considering only binary relations). However, we believe that these techniques can be extended for deciding containment of star join queries [RG02, GMUW08] under bag and bag-set semantics.

Chapter 4

View Selection under Bag and Bag-Set semantics

The hardness of the view selection, as we have seen in Section 2.5, is caused by the bicriteria nature of the problem. The appropriate order, however, of criteria may reduce significantly the space of candidate solutions. We remind, now, the two criteria; which are: (1) for a given set of views, the selection of the less-costly equivalent rewritings of the queries and (2) the choice of the appropriate set of views which does not violates the storage constraint. The observation that many equivalent rewritings of the same query may be produced by a given set of views [Hal01, ALM02] (but not the opposite) lead us to investigate the approach in which the criterion (2) takes place first.

In this chapter, we focus on both bag- and bag-set-oriented version of the view selection problem. For both versions, we limit the domain of the second criterion by imposing certain restrictions on the candidate views. More specifically, we describe a space of viewsets (constructed from the given query workload) which guarantees the existence of at least one optimal solution in the case that a solution for a given problem input exists. In addition, in the case of bag-set semantics, we formally describe the form of the views in bag-set equivalent rewritings.

Considering a view selection problem input (either bag- or bag-set-oriented), in Figure 4.1, we illustrate the space Ω of candidate viewsets (i.e., the set of views that may give equivalent rewritings of queries in the given workload). S_1 depicts the admissible viewsets. Further, S_2 depicts the set of optimal viewsets and S_3 depicts the sets of views whose body of definition is a generalization of a subexpression of the body of some query in the workload (see Section 2.5). The space of viewsets we propose is described by S_4; and is given by a set of solutions called *representative (optimal) set of solutions*.

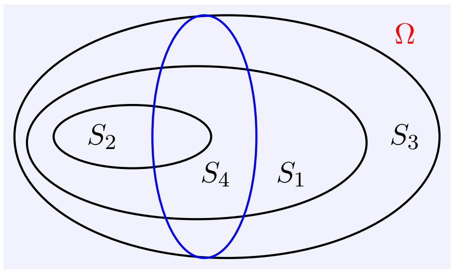


Figure 4.1: Space of viewsets

In order, now, to describe this space we elaborate on search space analysis of both bag and bag-set equivalent rewritings of CQs. In particular, we completely describe the form that each bag or bag-set equivalent rewriting has for a given CQ; and using this analysis we focus on the set of locally-minimal rewritings in order to construct the viewsets that solve the view selection problem. Practically, for a view selection problem input (either bag- or bag-set-oriented) the space S_4 (Figure 4.1) consists of viewsets constructed such that the following properties are satisfied:

- Each CQ in the given workload can be equivalently rewritten using each viewset in this space, and each view included in these viewsets can be used by an equivalent rewriting of a CQ in the given workload. This is achieved by constructing the viewsets from the expansions of equivalent rewritings of the CQs in the workload.

- Each admissible viewset for the given input which is not included in space S_4 can be produced from one or more viewsets included in S_4 by generalizing the corresponding views. This implies that the space S_4 contains the viewsets with minimum size for every database instance.

In addition, this approach is based on the observation that a single optimal solution is enough in order to solve the problem.

4.1 Space of optimal solutions under bag semantics

In this section, we elaborate on the search space analysis of candidate solutions for bag-oriented view selection problems, considering that both queries and views are conjunctive queries/views. Moreover, we consider monotonic cost models for computing the efficiency of the equivalent rewritings. The main results of this section are as follows:

1. In Subsections 4.1.1-4.1.3, we propose techniques to reduce the search space of candidate views and demonstrate that if there exists a solution for a given problem input, then there is at least one optimal solution of a specific form. We refer to these solutions as the *representative (optimal) set of solutions.*

2. In Subsection 4.1.4, an algorithm is presented that computes the representative set of optimal solutions.

4.1.1 On restricting the space of admissible viewsets

In Section 2.5, we noticed that considering bag semantics for workloads of conjunctive queries, each view in any admissible viewset (and thus in any optimal viewset) can be defined as a generalization of a subexpression of some query in the workload. Moreover, Lemmas 1 and 2 precisely describe a search space (consisting of all query subexpressions and their generalizations) to look for view definitions. As, in general, this search space is huge, it is crucial to investigate ways to reduce this search space (possibly for special cases of the view selection problem) in order to construct efficient algorithms for solving the view selection problem. A significant improvement in this direction might be to restrict the search space to contain only the subexpressions of the queries in the query workload (i.e. to exclude the generalizations of the subexpressions). Unfortunately, as it is shown in the following example, in the general case this is not possible.

Example 31. Consider a database schema $\mathcal{S}$ that contains only the relation e of arity 4 and a query workload $\mathcal{Q} = \{Q_1, Q_2\}$ on $\mathcal{S}$, where:

$$Q_1 : \; q_1(X, Y) \text{ :- } e(X, X, X, Y).$$
$$Q_2 : \; q_2(X, Y) \text{ :- } e(X, Y, Y, Y).$$

Consider also the following three viewsets $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$:

- $\mathcal{V}_1 = \{V_{11}, V_{12}\}$, where:

$$V_{11} : v_{11}(X_1, X_2) \text{ :- } e(X_1, X_1, X_1, X_2).$$
$$V_{12} : v_{12}(X_1, X_2) \text{ :- } e(X_1, X_2, X_2, X_2).$$

- $\mathcal{V}_2 = \{V_2\}$, where:
$$V_2 : v_2(X_1, X_2, X_3) \text{ :- } e(X_1, X_2, X_2, X_3).$$
- $\mathcal{V}_3 = \{V_3\}$, where:
$$V_3 : v_3(X_1, X_2, X_3, X_4) \text{ :- } e(X_1, X_2, X_3, X_4).$$

Notice that the bodies of the view definitions of $\mathcal{V}_1$ are subexpressions of the bodies of the queries in $\mathcal{Q}$ (in fact they are obtained from the bodies of Q_1 and Q_2 by renaming their variables), while the bodies of the views in $\mathcal{V}_2$ and $\mathcal{V}_3$ are generalizations of these subexpressions. Using each one of the above viewsets we get equivalent rewritings for the queries in $\mathcal{Q}$. More specifically, using $\mathcal{V}_1$ we get:
$$R_1 : r_1(X, Y) \text{ :- } v_{11}(X, Y).$$
$$R_2 : r_2(X, Y) \text{ :- } v_{12}(X, Y).$$
where R_1 and R_2 are equivalent rewritings of Q_1 and Q_2 respectively. Using $\mathcal{V}_2$ we get:
$$R'_1 : r'_1(X, Y) \text{ :- } v_2(X, X, Y).$$
$$R'_2 : r'_2(X, Y) \text{ :- } v_2(X, Y, Y).$$
where R'_1 and R'_2 are equivalent rewritings of Q_1 and Q_2 respectively. Finally, using $\mathcal{V}_3$ we get:
$$R''_1 : r''_1(X, Y) \text{ :- } v_3(X, X, X, Y).$$
$$R''_2 : r''_2(X, Y) \text{ :- } v_3(X, Y, Y, Y).$$
where R''_1 and R''_2 are equivalent rewritings of Q_1 and Q_2 respectively.

Assuming a database instance D={(e(a, a, a, a)), (e(a, b, c, d);5)}, the sets $\mathcal{V}_1(D)$, $\mathcal{V}_2(D)$ and $\mathcal{V}_3(D)$ are:
$$\mathcal{V}_1(D) = \{(v_{11}(a, a)), (v_{12}(a, a))\}.$$
$$\mathcal{V}_2(D) = \{(v_2(a, a, a))\}.$$
$$\mathcal{V}_3(D) = \{(v_3(a, a, a, a)), (v_3(a, b, c, d); 5)\}.$$
Since $size(\mathcal{V}_3(D)) = 6$, $size(\mathcal{V}_1(D)) = 2$ and $size(\mathcal{V}_2(D)) = 1$, we have $size(\mathcal{V}_3(D)) > size(\mathcal{V}_1(D)) > size(\mathcal{V}_2(D))$. If we choose a storage limit $L = size(\mathcal{V}_2(D)) = 1$, then $\mathcal{V}_2$ is the only admissible viewset among the above three. □

Example 31 shows that, in some cases, any optimal solution requires views that cannot be constructed as subexpressions of the queries in the query workload. The optimal solution in Example 31 uses views constructed using generalizations of subexpressions of the queries. In particular, the view in the optimal viewset $\mathcal{V}_2$ is defined as a common generalization of the bodies of both queries in the query workload $\mathcal{Q}$. Based on these observations two questions arise:

1. Are there any special cases of the view selection problem for which

there are optimal solutions whose viewset can be constructed by considering only subexpressions of the queries in the query workload?

2. For the general case, can we reduce the search space specified by Lemma 1 which consists of all possible generalizations of query subexpressions?

Both questions can be answered affirmatively as shown in the following Propositions 12 and 13.

Proposition 12. *Let $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ be a conjunctive bag-oriented view selection problem input such that every relation in $\mathcal{S}$ appears at most once in a body of some query in $\mathcal{Q}$. If there exists a solution for $\mathcal{P}$, then there exists an optimal solution $\Lambda = (\mathcal{V}, \mathcal{R})$ such that each view in $\mathcal{V}$ is defined as a subexpression of a query in $\mathcal{Q}$.*

Proof. Considering that there is a solution for $\mathcal{P}$, we have that there is also an optimal solution $\Lambda_o = (\mathcal{V}_o, \mathcal{R}_o)$ for $\mathcal{P}$. Without lost of generality we consider that for each CQ in $\mathcal{Q}$ there is a unique optimal rewriting in $\mathcal{R}_o$. We will construct a viewset $\mathcal{V}$ from $\mathcal{V}_o$, without losing the existence of optimal rewritings. In addition, the construction of $\mathcal{V}$ will guarantee that the body of each view in $\mathcal{V}$ is a subexpression of a query in $\mathcal{Q}$. Then, we will show that the previous constructed viewset is admissible by proving that it does not violates the storage limit constraint.

We construct a viewset $\mathcal{V}$ from $\mathcal{V}_o$ as follows. For every view $V_o \in \mathcal{V}_o$ for which there is a unique V_o-subgoal in a body of a rewriting in $\mathcal{R}_o$, we add into $\mathcal{V}$ the view V whose definition is the view-expansion of this view-subgoal. By Lemma 1, however, we conclude that the body of V_o is a generalization of subexpression of a body of a CQ in $\mathcal{Q}$. Since, now, every relation in $\mathcal{S}$ appears at most once in a body of some query in $\mathcal{Q}$, we conclude that V_o is used by a single optimal rewriting of $\mathcal{Q}$, and there is only one view-subgoal of this rewriting that refers to V_o. Hence, for each view in $\mathcal{V}_o$ there is a unique view in $\mathcal{V}$ which is constructed as we previously described.

In addition, we construct the set of rewritings $\mathcal{R}$ as follows. For each rewriting R_o in $\mathcal{R}_o$ we replace each view-subgoal g with $\theta(v)$, where v is the head of the definition of the corresponding view V in $\mathcal{V}$ and θ is a renaming substitution over the definition of V such that $body(\theta(V))$ is identical to the body of view-expansion of g. Each rewriting R obtained by this procedure is included in $\mathcal{R}$. By the construction of $\mathcal{V}$ and $\mathcal{R}$, we easily conclude that there is at least one bag equivalent rewriting of each query in $\mathcal{Q}$ using $\mathcal{V}$.

In order, now, to prove that $\Lambda = (\mathcal{V}, \mathcal{R})$ is an optimal solution for $\mathcal{P}$, we have to prove that (1) $\mathcal{V}$ is admissible (i.e., satisfies the storage constraint L) and (2) each rewriting $R \in \mathcal{R}$ is optimal.

The first assumption can be proved by the construction of $\mathcal{V}$; since for each view V in $\mathcal{V}$ and the corresponding view V_o in $\mathcal{V}_o$ we have that V_o is a generalization of V_o. Hence, there is a subgoals-onto containment mapping from V_o' to V'; which implies that $V_o \sqsubseteq_b V$ (Proposition 4). Thus, the size of $\mathcal{V}$ is at most equal to the size of $\mathcal{V}_o$ (i.e., for every $\mathcal{D}$ of $\mathcal{S}$, we have $size(\mathcal{V}(\mathcal{D})) \leq L$); namely, $\mathcal{V}$ is admissible.

The second assumption is proved, as follows. For each rewriting $R \in \mathcal{R}$ and the corresponding rewriting $R_o \in \mathcal{R}_o$, the following happens: As R_o is an optimal rewriting of a query $Q \in \mathcal{Q}$, then there is an optimal query plan for R_o. Let this optimal plan be the following:

$$(\ldots((g_1 \bowtie g_2) \bowtie g_3) \bowtie \ldots \bowtie g_m),$$

where for each $i = 1, \ldots, m$, g_i is a reference to a view in $\mathcal{V}_o$, m is the number of subgoals of R_o. Now, by construction of R we have the following query plan:

$$(\ldots((\theta_1(v_1) \bowtie \theta_2(v_2)) \bowtie \theta_3(v_3)) \bowtie \ldots \bowtie \theta_m(v_m)),$$

where for each $i = 1, \ldots, m$, the view-subgoal $\theta_i(v_i)$ of R replaced the view-subgoal of R_o, during the construction of R (as previously described). Comparing, now, the above two query plans we have that the size of g_i is greater than or equal to $\theta_i(v_i)$ (as previously proved); hence, by construction of R we have that the size of each intermediate relation of the second query plan is less than or equal to the corresponding intermediate relation of the first query plan. Thus, as we assumed monotonic cost models, we have that R is as efficient as R_o. Consequently, by definition of optimal solution, Λ is an optimal solution for $\mathcal{P}$. $\square$

Example 32. Consider a database schema $\mathcal{S}$ that contains the binary relations r_1, r_2, r_3 and r_4 and a query workload $\mathcal{Q} = \{Q_1, Q_2\}$ on $\mathcal{S}$, where:

$$Q_1: \; q_1(X,Y) \text{ :- } r_1(X,X), r_2(X,Y).$$
$$Q_2: \; q_2(X,Y) \text{ :- } r_3(X,Z), r_4(Z,Y).$$

In addition, all the views that can be used by a bag equivalent rewriting of a CQ in $\mathcal{Q}$ have the following definitions (notice that using arbitrary renaming substitutions we can find similar view definitions that can also be used by a bag equivalent rewriting).

$V_1: \ v_1(X,Y) \text{ :- } r_1(X,X), r_2(X,Y).$
$V_2: \ v_2(X,Z,Y) \text{ :- } r_1(X,Z), r_2(Z,Y).$
$V_3: \ v_3(X,Z,W,Y) \text{ :- } r_1(X,Z), r_2(W,Y).$
$V_4: \ v_4(X) \text{ :- } r_1(X,X).$
$V_5: \ v_5(X,Y) \text{ :- } r_1(X,Y).$
$V_6: \ v_6(X,Y) \text{ :- } r_2(X,Y).$
$V_7: \ v_7(X,Y) \text{ :- } r_3(X,Z), r_4(Z,Y).$
$V_8: \ v_8(X,Z,Y) \text{ :- } r_3(X,Z), r_4(Z,Y).$
$V_9: \ v_9(X,Z,W,Y) \text{ :- } r_3(X,Z), r_4(W,Y).$
$V_{10}: \ v_{10}(X,Y) \text{ :- } r_3(X,Y).$
$V_{11}: \ v_{11}(X,Y) \text{ :- } r_4(X,Y).$

Suppose, now, the database $\mathcal{D} = \{r_1(a,a),\ r_2(a,a),\ r_3(a,a),\ r_3(b,c),\ r_4(a,a)\}$ and the storage limit $L = 2$ tuples. It is easy to verify that the admissible viewsets for the bag-oriented problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$, contain one of the first three views, and either V_7 or V_8 (any other combination of the above views either violates the storage limit or does not give bag equivalent rewriting for both queries). Moreover, notice that all views included in an admissible viewset for $\mathcal{P}$ are defined as generalization of subexpressions of queries' bodies. For instance, consider the viewset $\mathcal{V} = \{V_1, V_7\}$. It is easy to see, moreover, that $\mathcal{V}$ is an optimal viewset for $\mathcal{P}$. □

Notice that, when the assumptions of Proposition 12 hold, the queries in the workload $\mathcal{Q}$ do not contain self-joins. In this case, as referred in Section 2.5, we can rewrite each query in $\mathcal{Q}$ without using self-joins of views in $\mathcal{V}$.

We now turn our attention to the general case and prove that, in order to construct an optimal viewset, we need to consider both subexpressions of queries and lgg"s of subexpressions. We can thus exclude all those generalizations of subexpressions that are not lgg"s of two or more subexpressions. This is made formal in the following proposition:

Proposition 13. *Let $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ be a conjunctive bag-oriented view selection problem input. If there exists a solution for $\mathcal{P}$, then there is an optimal solution $\Lambda = (\mathcal{V}, \mathcal{R})$ for $\mathcal{P}$ such that the body of each view in $\mathcal{V}$ is either a subexpression of the body of a query in $\mathcal{Q}$ or an lgg of two or more subexpressions of the bodies of queries in $\mathcal{Q}$.*

Proof. Considering that there is a solution for $\mathcal{P}$, we have that there is also an optimal solution $\Lambda_o = (\mathcal{V}_o, \mathcal{R}_o)$ for $\mathcal{P}$. This proof is similar to the proof of Proposition 12.

We construct a viewset $\mathcal{V}$ from $\mathcal{V}_o$ as follows. For every view $V_o \in \mathcal{V}_o$ for which there is a single V_o-subgoal in a body of a unique rewriting in $\mathcal{R}_o$, we add into $\mathcal{V}$ the view V whose definition is the view-expansion of this view-subgoal. On the other hand, for each view $V_o \in \mathcal{V}_o$ for which multiple V_o-subgoals appear in rewritings in $\mathcal{R}_o$, we add into $\mathcal{V}$ the view V whose definition is the lgg of all view-expansions of the V_o-subgoals.

In addition, we construct the set of rewritings $\mathcal{R}$ as follows. For each rewriting R in $\mathcal{R}$ we replace each view-subgoal v with $\theta(v')$, where v' is the head of the definition of the corresponding view in $\mathcal{V}$ and θ is the appropriate substitution such that the bodies of the view-expansions of v and definition of $\theta(v')$ are the same expression. By the construction of $\mathcal{V}$ and $\mathcal{R}$, we conclude that there is at least one bag equivalent rewriting of each query in $\mathcal{Q}$ using $\mathcal{V}$.

In order, now, to prove that $\Lambda = (\mathcal{V}, \mathcal{R})$ is an optimal solution for $\mathcal{P}$, we have to prove that (1) $\mathcal{V}$ is admissible (i.e., satisfies the storage constraint L) and (2) each rewriting $R \in \mathcal{R}$ is optimal.

The first assumption can be proved by the construction of $\mathcal{V}$. To verify this notice that as for each view V in $\mathcal{V}$ and the corresponding view V_o in $\mathcal{V}_o$ we have that V_o is a generalization of V. Hence, there is a subgoals-onto containment mapping from V_o' to V'; which implies that $V_o \sqsubseteq_b V$ (Proposition 4). Thus, the size of $\mathcal{V}$ is at most equal to the size of $\mathcal{V}_o$ (i.e., for every $\mathcal{D}$ of $\mathcal{S}$, we have $size(\mathcal{V}(\mathcal{D})) \leq L$); namely, $\mathcal{V}$ is admissible.

The proof of the second assumption is identical to the proof of the second assumption appearing in the proof of Proposition 12, and based on the constructions of $\mathcal{V}$ and $\mathcal{R}$. Consequently, by definition of optimal solution, Λ is an optimal solution for $\mathcal{P}$. $\square$

The intuition behind Propositions 12 and 13 is that the use of generalization of subexpressions in defining a view is useful only when this view definition will be subsequently used two or more times to construct equivalent rewritings for the queries in the workload $\mathcal{Q}$. This is the case of the viewsets $\mathcal{V}_2$ and $\mathcal{V}_3$ in Example 31. Besides, it is not useful to generalize the subexpression more than needed as this, in general, increases the number of the tuples obtained when materializing this "overgeneralized" view definition and this does not contribute towards an improvement of the evaluation of the rewriting. An example of such "overgeneralization" is the viewset $\mathcal{V}_3$ in Example 31.

4.1.2 Minimum set of distinguished variables in candidate rewritings

In this section, we further refine Propositions 12 and 13 by restricting also the vector of variables in the heads of the view definitions. The simplest choice is to put as arguments of a view head all different variables appearing in the view's body. However, this is not always the "best" choice as the following example shows.

Example 33. Consider a query workload $\mathcal{Q} = \{Q\}$, where:

$$Q:\ q_1(X,Y) \text{ :- } e(X,Z), f(Z,W), g(W,Y).$$

Consider also the following viewset $\mathcal{V}_1 = \{V_{11}, V_{12}\}$:

$$V_{11} : v_{11}(X,Z,W) \text{ :- } e(X,Z), f(Z,W).$$
$$V_{12} : v_{12}(W,Y) \text{ :- } g(W,Y).$$

Notice that using $\mathcal{V}_1$ as we can get the following equivalent rewriting R of Q:

$$R : r(X,Y) \text{ :- } v_{11}(X,Z,W), v_{12}(W,Y).$$

It is easy to see, however, that the variable Z in the head of V_{11} is redundant. More specifically, if we replace the view V_{11} in $\mathcal{V}_1$ by the following view V'_{11}:

$$V'_{11} : v'_{11}(X,W) \text{ :- } e(X,Z), f(Z,W).$$

we get R' which is also an equivalent rewriting of Q:

$$R' : r'(X,Y) \text{ :- } v'_{11}(X,W), v_{12}(W,Y).$$

Comparing V_{11} and V'_{11}, it is easy to see that, under bag semantics, for every database D we have $size(\mathcal{V}_{11}(D)) = size(\mathcal{V}'_{11}(D))$. Also, the query R', obtained by using V'_{11} to rewrite Q, is computed more efficiently than the rewriting R obtained by using V_{11} to rewrite Q. □

We now show how to choose the appropriate set of variables to be used as head arguments of the view definitions. In order to find this set we define the notion of *linking variables* of a CQ and a subexpression of its body. The linking variables are related to the shared-variables property introduced by [PH01]; that holds in the set-oriented context.

Definition 11. Let Q be a query of the form H :- $B_1, \ldots, B_n$ and $S = B_{11}, \ldots, B_{1k}$, with $1 \le k \le n$, be a subexpression of the body of Q. Let $Q' = Q - S$ be the query obtained by removing from the body of Q the atoms in S. Then, the set $lvars(Q,S) = vars(Q') \cap vars(S)$, is called the *linking variables* of Q and S.

Example 34. (Continued from Example 33) Consider the query Q in Example 33 and the subexpression $S = e(X,Z), f(Z,W)$ of Q. It is easy to see that the set of linking variables of Q and S is $lvars(Q,S) = \{X,W\}$. □

Using now the linking variables of a CQ Q and a subexpression S of its body, we describe the minimum set of required distinguished variables of a view V such that V covers S in a bag equivalent rewriting of Q which uses V. The following proposition formally describes this set by considering that V is a generalization of S.

Proposition 14. *Let Q be a CQ and V be a view whose body is defined as a generalization of a subexpression S of Q. Then V covers S in a bag equivalent rewriting of Q using V,* ***if and only if*** *there is a substitution θ over V such that the following hold:*

1. *the body of $\theta(V)$ is identical to S,*
2. *for each variable X of V such that $\theta(X)$ is contained in $lvars(Q, S)$, X is a distinguished variable of V; and*
3. *for every two variables Y, Z of V such that $Y \neq Z$ and $\theta(Y) = \theta(Z)$, Y and Z are distinguished variables of V.*

Proof.

(If part) Consider the view V' which obtained as follows: the body of V' is the subexpression $S' = body(Q) - S$, and all variables appearing in its body are distinguished variables. We construct, now, the query R such that its head is identical to the head of Q and its body contains the atoms $head(\theta'(V'))$ and $head(\theta(V))$, where θ' is a substitution over V' such that for each variable X of V' we have that if X is contained in $lvars(Q, S')$ then $\theta'(X) = X$; otherwise $\theta'(X)$ is a fresh variable. By construction of V' and R, we conclude that $R^{exp} \equiv_b Q$.

(Only-If part) Let, now, the bag equivalent rewriting R of Q using V such that V covers S. By Proposition 8 and Lemma 2, we conclude that there is a renaming substitution ϕ over R^{exp} such that $\phi(R^{exp}) \equiv_b Q$ (especially, $\phi(R^{exp})$ and Q are identical); therefore there is a substitution ϕ' over V such that $body(\phi(\phi'(V))) = S$. We refer to the substitution over V implied by the composition of ϕ' and ϕ as θ (i.e., $\theta = \phi \circ \phi'$). Hence, θ satisfies the condition 1.

(θ satisfies condition 2) Consider, now, an arbitrary variable X of V such that $\theta(X)$ is in $lvars(Q, S)$. Suppose that X is included in $NDVars(V)$. By definition of linking variables, $\theta(X)$ appears both in $vars(S)$ and in $vars(\phi(R^{exp}) - S)$. However, by definition of expansion, $\theta(X)$ is a fresh variable; which contradicts with the assumption that ϕ is a renaming substitution. Hence, X is a distinguished variable of V.

(θ satisfies condition 3) Consider, now, two variables Y, Z of V such that $Y \neq Z$ and $\theta(Y) = \theta(Z)$. Moreover, suppose that at least one of Y and Z is included in $NDVars(V)$. Without loss of generality, let Y in $NDVars(V)$. By definition of expansion, we have that $\theta(Y)$ is a fresh variable; i.e., $\theta(Y) \neq \theta(Z)$. However, this is a contradiction. Hence, both Y and Z are distinguished variables of V. □

Since the body of each view which is used in a bag equivalent rewriting is a generalization of a subexpression of query's body, the previous proposition determines the minimum set of variables that should be put in the head of definition of each view so as the view can be used by a bag equivalent rewriting of the query. Here, we distinguish the two cases introduced by Proposition 13; (1) when the body of the view is defined as subexpression of query's body and (2) when the body of the view is defined as lgg of subexpressions (either of the same query's body or of different bodies).

In the case (1) the Proposition 14 indicates the set of linking variables is the minimum set of variables that should be put in the head of the view definition so as this view can be used in a bag equivalent rewriting of the query. The following example illustrates such a case.

Example 35. (Continued from Example 34) Notice that the variables in $\{X, W\}$ (which are the linking variables of Q and S) appearing in the heads of both views V_{11} and V'_{11} are constructed from the subexpression S of Q. Observe that if we remove either X or W, or both X and W, from the head of the view V_{11} (or the view V'_{11}), then the corresponding viewset cannot give equivalent rewriting for the query Q. □

Considering, now, the case (2) the body of the view may be either identical to some subexpressions of queries bodies (up to a renaming substitution) or a proper lgg of them (i.e., lgg which is not identical to every subexpression). In the case that the body is a proper lgg the sets of linking variables do not suffice in order to determine the head variables of the view definition. In particular, the Proposition 14 shows that in this case the linking variables of the query and each subexpression that the certain view covers, do not determine the minimum set of variables that must be put in the head of view. The third condition of the Proposition 14 describes the additional distinguished variables. The following example illustrates such a case.

Example 36. Consider a query workload $\mathcal{Q} = \{Q_1, Q_2\}$, where:
$Q_1 :\ q_1(X_1, X_4)$:- $e(X_1, X_1, X_1, X_2, X_3), f(X_3, X_2, X_4)$.
$Q_2 :\ q_2(Y_1, Y_2, Y_7)$:- $e(Y_1, Y_2, Y_2, Y_3, Y_4), f(Y_4, Y_5, Y_6), g(Y_6, Y_7)$.

Consider also the following viewset $\mathcal{V}_1 = \{V_1, V_2\}$, where:

$V_1 : v_1(Z_1, Z_2, Z_3, Z_5, Z_6) \text{ :- } e(Z_1, Z_2, Z_2, Z_3, Z_4), f(Z_4, Z_5, Z_6).$

$V_2 : v_2(Z_6, Z_7) \text{ :- } g(Z_6, Z_7).$

Notice that body of the view V_1 is obtained as the lgg of the subexpressions S_1 and S_2 of Q_1 and Q_2 respectively, where:

$$S_1 = e(X_1, X_1, X_1, X_2, X_3), f(X_3, X_2, X_4).$$
$$S_2 = e(Y_1, Y_2, Y_2, Y_3, Y_4), f(Y_4, Y_5, Y_6).$$

Moreover, using the substitutions $\theta_1 = \{Z_1/X_1,\ Z_2/X_1,\ Z_3/X_2,\ Z_4/X_3,\ Z_5/X_2,\ Z_6/X_4\}$ and $\theta_2 = \{Z_1/Y_1,\ Z_2/Y_2,\ Z_3/Y_3,\ Z_4/Y_4,\ Z_5/Y_5,\ Z_6/Y_5\}$ over V_1 we conclude that S_1 and S_2, respectively, are images of $body(V_1)$.

The linking variables of S_1 and Q_1 are $lvars(Q_1, S_1) = \{X_1, X_4\}$, while the linking variables of S_2 and Q_2 are $lvars(Q_2, S_2) = \{Y_1, Y_2, Y_6\}$.

Consider, now, the set of body variables of V_1 that map on the corresponding linking variables of Q_1, Q_2 and S_1, S_2, respectively; which contain the variables Z_1, Z_2, Z_5 and Z_6. We can easily verify that although the variable Z_3 is not included in this set it must be put in the head of V_1 in order to exists a bag equivalence rewriting of Q_2 using V_1. Z_3 is put in the head according to the third condition of Proposition 14. Therefore the set of the variables in the head of the view V_1 is $\{Z_1,\ Z_2,\ Z_3,\ Z_5,\ Z_6\}$.

Notice that using the above viewset we obtain the following rewritings R_1 and R_2 for Q_1 and Q_2, respectively. Notice that using θ_1 and θ_2 the view V_1 covers S_1 and S_2, respectively:

$$R_1 :\ r_1(X_1, X_4) \text{ :- } v_1(X_1, X_1, X_2, X_2, X_4).$$
$$R_2 :\ r_2(Y_1, Y_2, Y_7) \text{ :- } v_1(Y_1, Y_2, Y_3, Y_5, Y_6), v_2(Y_6, Y_7).$$

□

4.1.3 Representative set of solutions under bag semantics

Summarizing the results of two previous sections, we describe how we can construct viewsets that solve the bag-oriented view selection problem. This is achieved by describing the way that each view definition in such a viewset is constructed.

Consider, now, a query workload $\mathcal{Q}$ over a schema $\mathcal{S}$ and three subexpressions S, S_1 and S_2 of the bodies of queries in $\mathcal{Q}$ such that there is an lgg of S_1 and S_2. Using the Proposition 14, we construct a view V whose body is defined as the subexpression S (also called *subexpression view*) as follows: the body of V is identical to S and its head contains the linking variables of the CQ of $\mathcal{Q}$ that contains S, and S. In addition, we can construct the view V' whose body is the least general generalization of S_1 and S_2 using the following steps:

1. We construct the views V_1 and V_2 using the subexpressions S_1 and S_2 respectively as bodies, and we put as head variables the linking variables of the corresponding queries in $\mathcal{Q}$ that contain S_1, S_2, and S_1, S_2, respectively.

2. By considering V_1 and V_2 as queries we construct V' with body the lgg of the bodies of V_1 and V_2. Moreover, using Proposition 14, we put in the head of V' the minimum required set of variables such that treating both V_1 and V_2 as queries, V' covers both the bodies of V_1 and V_2. V' is said to be an *lgview* of the views V_1 and V_2; and denoted as $lgview(V_1, V_2)$.

An example of an lgview was given in Example 36, where V_1 is an lgview. This procedure can be easily generalized for more than two subexpressions.

An interesting question referring to lgviews is the following: "Does the inequality $size(V) \leq size(V_1) + size(V_2)$ always hold for the lgview V of two views V_1 and V_2?". Notice that, if the answer is "yes" for any bag-oriented view selection problem input, then whenever an lgview exists, the original views can be discarded eliminating in this way the search space for finding viewsets. Unfortunately, the inequality does not always hold, as the following example shows.

Example 37. Let a viewset $\mathcal{V} = \{V_1, V_2\}$, where the definitions of the views are:

$$V_1 : v_1(X,Z) \text{ :- } p_1(X,X), p_2(X,Z).$$
$$V_2 : v_2(X,Z) \text{ :- } p_1(X,Z), p_2(Z,Z).$$

where p_1 and p_2 are binary relations on the database schema $\mathcal{S}$. Consider also another viewset $\mathcal{W} = \{W\}$ whose view W is defined as:

$$W : w(A,B,C) \text{ :- } p_1(A,B), p_2(B,C).$$

Notice that W is the lgview of the views in $\mathcal{V}$. Assuming the database instance:

$$\mathcal{D} = \{p_1(1,1), p_1(1,2), p_1(3,4), p_2(1,1), p_2(1,2), p_2(2,2), p_2(2,3), p_2(4,5)\},$$

in which the multiplicity of each database tuple in this example is 1 and for this we omit it, and materializing the views over this database we get:

$$\mathcal{V}(\mathcal{D}) = \{v_1(1,1), v_1(1,2), v_2(1,1), v_2(1,2)\}.$$
$$\mathcal{W}(\mathcal{D}) = \{w(1,1,1), w(1,1,2), w(1,2,2), w(1,2,3), w(3,4,5)\}.$$

It is easy to see that $\text{size}(\mathcal{V}(\mathcal{D})) < \text{size}(\mathcal{W}(\mathcal{D}))$. □

The following theorem summarizes the results given by Proposition 13 and Proposition 14, and shows that if there is a solution for a given bag-oriented problem input then there is an optimal viewset that contains only subexpression views and lgviews.

Theorem 15. *Let a bag-oriented view selection input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$. If there is a solution for $\mathcal{P}$, then there exists an optimal solution $\Lambda = (\mathcal{V}, \mathcal{R})$ such that each view in $\mathcal{V}$ is either a subexpression view or an lgview of subexpression views of queries in $\mathcal{Q}$.*

Thus the class of solutions constructed as above is a *representative set of solutions* for a given bag-oriented view selection problem input $\mathcal{P}$.

4.1.4 LGG-VSB Algorithm

An algorithm, called LGG-VSB, which is based on the results of the previous section, and outputs the representative set of optimal solutions, for a given bag-oriented view selection problem input, is proposed in this section. LGG-VSB incorporates the results of the Theorem 15 and Lemma 2 to the algorithm CGALG (introduced in [ACGP07]), reducing significantly the search space for finding an optimal solution. In particular, LGG-VSB avoids the construction of viewsets that do not rewrite the queries in the workload, by producing the candidate viewsets in such a way that the construction of the equivalent rewritings of the query is quickly achieved; i.e. instead of construction of every set of views whose body is a generalization of a subexpression of a query's body (CGALG), LGG-VSB constructs viewsets that form a partition of the body of each query in the workload.

Algorithm *LGG-VSB.*

```
  Input: A bag oriented view selection problem input[1] $\mathcal{P} = \{\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L}\}$.
  Output: $\Lambda$, the representative set of optimal solutions.
Begin
   1. Let $\mathcal{V}$ be a set of viewsets constructed as follows: Each $\mathcal{V}' \in \mathcal{V}$ is of
     the form $\mathcal{V}' = \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_n$, where $n$ is the number of queries in $\mathcal{Q}$
      and each viewset $\mathcal{V}_i$ is obtained from the query $Q_i \in \mathcal{Q}$ as follows:
       - Let $P_i$ be a partition of the subgoals of $Q_i$.
       - For each block $B_j \in P_i$, add a view definition $V_{i,j}$ in $\mathcal{V}_i$ whose body
          consists of the atoms in $B_j$ and whose head variables are the
          variables in $lvars(Q_i, B_j)$.
   2. Set $G_0 = \mathcal{V}$; set $i = 0$.
   3. while $G_i \neq \emptyset$ do
       - $G_{i+1} = \{\mathcal{V}_g | \mathcal{V}_g = (\mathcal{V}' - \mathcal{M}) \cup \{V_l\}$, where $\mathcal{V}' \in G_i$ and $\mathcal{M} \subseteq \mathcal{V}'$
            and $V_l = lgview(\mathcal{M})\}$.
       - i = i + 1.
```

[1]Recall that $\mathcal{L} = \{L\}$, where L is a single storage limit constraint.

```
    end while
  4. Let $\mathcal{V} = \bigcup_{j=0,\ldots,i} G_j$.
  5. Compute the cost $C(\mathcal{Q},\mathcal{D})$ of $\mathcal{Q}$ on $\mathcal{D}$ and set it to $C_{opt}$.
  6. For every viewset $\mathcal{V}' \in \mathcal{V}$, such that $size(\mathcal{V}') \leq L$, do
    - Construct the set $\mathcal{R}_{\mathcal{V}'}$ of all equivalent rewritings of $\mathcal{Q}$ using $\mathcal{V}'$.
    - Set $\Lambda = \emptyset$.
    - For every distinct subset $\mathcal{R}$ of $\mathcal{R}_{\mathcal{V}'}$ such that $\mathcal{R}$ contains an
        equivalent rewriting of each query in $\mathcal{Q}$, do
      - Let $c = C(\mathcal{R}, \mathcal{V}'(\mathcal{D}))$.
      - If $c < C_{opt}$, then set $C_{opt} = c$ and set $\Lambda = \{(\mathcal{V}', \mathcal{R})\}$
        else if $c = C_{opt}$, then $\Lambda = \Lambda \cup \{(\mathcal{V}', \mathcal{R})\}$.
end.
```

4.2 Space of Optimal solutions under bag-set semantics

In this section, we focus on the bag-set-oriented view selection problem where queries, rewritings and views are defined as CQs. Although, under bag semantics, the form of each view that can be used by an equivalent rewriting is completely defined by Lemmas 1, 2, and Proposition 14, under bag-set semantics, there is not any technique even for finding a bag-set rewriting using a given set of views (in the case of CQs).

Since, now, the bag-set semantics can be regarded as a special case of bag semantics, one may think that we can use the space of viewsets described by the Theorem 15 in order to solve the bag-set-oriented view selection problem (i.e., to find at least one optimal solution in the case that a solution for the given input exists). However, as the following example describes, there is a bag-set-oriented problem input such that even the space of viewsets described by view definitions whose bodies are generalizations of subexpressions of the given queries' bodies does not suffice in order to find an optimal solution.

Example 38. Consider the bag-set view selection problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, L)$ such that $\mathcal{S}$ contains the relation e of arity 7, $\mathcal{Q} = \{Q_1, Q_2\}$, the storage limit L is equal to 3 tuples, and the database $\mathcal{D}$ contains the tuples: $e(a,a,b,c,c,d,d)$, $e(a,b,b,e,f,d,g)$, $e(a',b',b',c',c',d',d')$, $e(a',a',b',e',f',d',g')$, e(o,o,o,o,o,o,o) and $e(x,x,x,y,z,w,u)$. The definitions of queries are the following:
$Q_1 : q_1(A,B,C,D,E,F,G)$:- $e(A,A,B,C,C,D,D), e(A,B,B,E,F,D,G)$.
$Q_2 : q_2(A,B,C,D,E,F,G)$:- $e(A,B,B,C,C,D,D), e(A,A,B,E,F,D,G)$.

Evaluating both queries on the database $\mathcal{D}$ we have that $Q_1(\mathcal{D}) = \{q_1(a,b,c,d,e,f,g), q_1(o,o,o,o,o,o,o)\}$ and $Q_2(\mathcal{D}) = \{q_2(a',b',c',d',e',f',g'), q_2(o,o,o,o,o,o,o)\}$.

Searching for admissible viewsets notice that there is not any admissible viewset for $\mathcal{P}$ that contains only subexpression views. On the other hand, consider the following two views.
$V_1 : v_1(U_1,\ldots,U_9)$:- $e(U_1,U_2,U_3,U_4,U_4,U_5,U_5), e(U_1,U_6,U_3,U_7,U_8,U_5,U_9)$.
$V_2 : v_2(T_1,\ldots,T_9)$:- $e(T_1,T_1,T_2,T_3,T_4,T_5,T_6), e(T_1,T_2,T_2,T_7,T_8,T_5,T_9)$.

Notice that both V_1 and V_2 are lgviews of queries' bodies and give bag-set equivalent rewritings of both Q_1 and Q_2. However, evaluating under bag-set semantics both V_1 and V_2 on $\mathcal{D}$ we can easily verify that $V_1(\mathcal{D})$ contains 5 tuples and $V_2(\mathcal{D})$ contains 4 tuples; hence none of them gives an admissible viewset for $\mathcal{P}$. Consequently, there is not any optimal solution that can be constructed as Theorem 15 specifies. On the other hand, considering the viewset $\mathcal{V} = \{V\}$, where V has definition:

$$\begin{aligned} V : v(X_1,X_2,Y_1,Y_2,Y_3,W_1,W_2,W_3,W_4,W_5,W_6,Z_1,Z_2,Z_3) \text{ :- }\\ e(X_1,X_1,X_2,W_1,W_2,Z_1,Z_2),\\ e(X_1,X_2,X_2,W_3,W_4,Z_1,Z_3),\\ e(Y_1,Y_2,Y_3,W_5,W_6,Z_1,Z_1).\end{aligned}$$

we can easily notice that $\mathcal{V}$ is an optimal viewset for $\mathcal{P}$; since $V(\mathcal{D})$ contain 3 tuples and gives bag-set equivalent rewritings for both Q_1 and Q_2. More specifically, R_1 and R_2 are the bag-set equivalent rewritings of Q_1 and Q_2, respectively, using $\mathcal{V}$, and have the following definitions.
$R_1 : q_1(A,B,C,D,E,F,G)$:- $v(A,B,A,A,B,C,C,E,F,C,C,D,D,G)$
$R_2 : q_2(A,B,C,D,E,F,G)$:- $v(A,B,A,B,B,E,F,C,C,C,C,D,G,D)$.

However, V is neither a subexpression view nor a lgview of subexpressions of queries' bodies. Moreover, notice that V is not even a generalization of any subexpression of a query's body. □

In this section, we firstly describe the search space of solutions for a given bag-set-oriented problem input by specifying the form of each view that can be used in a bag-set equivalent rewriting of a CQ (Section 4.2.1). Moreover, we describe the form of every bag-set equivalent rewriting of a given CQ and the combination of the views that can be used to equivalently rewriting a CQ under bag-set semantics. Furthermore, we determine a space of viewsets that guarantees the existence of an optimal viewset (when a solution for the given input exists); similar to that proposed for bag semantics in the previous section.

4.2.1 Useful viewsets for rewriting CQs under bag-set semantics

As it is mentioned in Sections 2.3.2 and 2.4, the existence of an isomorphic contaiment mapping between a CQ and the expansion of a rewriting of the CQ decides that the rewriting is bag-set equivalent only if both CQ and the expansion of the rewriting are duplicate-free (i.e., their definitions do not have duplicate subgoals). In particular, duplicate subgoals of either CQ or the expansion of the rewriting do not affect the existence of bag-set equivalent rewriting. Therefore, constructing from a CQ Q, the CQ Q^m by eliminating duplicate subgoals from Q, the canonical representation of each view-expansion of a view-subgoal in a bag-set equivalent rewriting of Q has body which is a subexpression of Q^m. This corollary is formally described as follows.

Corollary 4. *Let a CQ Q and a bag-set equivalent rewriting R of Q using a viewset $\mathcal{V}$. Then the following hold:*

1. *for every view-subgoal v of R the body of the canonical representation of the view-expansion of v is a subexpression of the body of the canonical representation of Q; and*

2. *there is a subset A of the subgoals of R that cover all the subgoals of Q and none view-subgoal in this subset is redundant (i.e., we cannot drop any view-subgoal from A and still cover all subgoals of Q).*

Consequently, the body of view-expansion of v can be an expression obtained by adding duplicate subgoals to the corresponding subexpression S of the body of canonical representation of Q. Moreover, notice that S can also be obtained by eliminating duplicates subgoals from a subexpression of Q. Hence, the body of view-expansion of v is obtained by a combination of additions and deletions of duplicate subgoals from S. The following definition captures such additions and deletions of duplicate subgoals in a CQ.

Definition 12. Considering two CQs Q_1 and Q_2, we say that Q_2 is a **duplicate-extension** of Q_1 if Q_2 can be obtained by a sequence of additions and deletions of duplicate subgoals from the body of Q_1.

Similarly, we say that an expression E_2 is a *duplicate-extension* of an expression E_1 if E_2 can be obtained by a sequence of additions and eliminations of duplicate atoms from E_1.

Since, now, the view-expansion of a view-subgoal of a rewriting of a CQ is obtained by applying a substitution over the definition of the view-subgoal, we conclude that the body of each view that can be used in a bag-set equivalent rewriting of a CQ is a generalization of a duplicate-extension of a subexpression of the CQ. This is formally described by the following proposition.

Proposition 15. *Let a CQ Q and a viewset $\mathcal{V}$. If there is a bag-set equivalent rewriting R of Q using $\mathcal{V}$ then the body of each view V of $\mathcal{V}$ used in R is a generalization of a duplicate-extension of a subexpression of Q's body.*

Proof. This proof immediately follows from Proposition 7 and the definitions of expansion and duplicate-extension. □

Example 39. (Continued from Example 38) Considering the bodies S_1 and S_2 of the queries Q_1 and Q_2, respectively, we can easily notice that both V_1 and V_2 are generalizations of duplicate-extensions of S_1 and S_2. Moreover, notice that the body of V is a common generalization of the following expressions:

$S'_1 = e(A, A, B, C, C, D, D), e(A, B, B, E, F, D, G), e(A, A, B, C, C, D, D)$.

$S'_2 = e(A, B, B, C, C, D, D), e(A, A, B, E, F, D, G), e(A, B, B, C, C, D, D)$.

In addition, notice that the expression S'_1 and S'_2 are duplicate-extensions of S_1 and S_2, respectively (both are obtained by adding one copy of the first subgoal of S_1 and S_2). □

Concerning now the minimum set of required variables in the head of a view definition such that the view can be used by a bag-set equivalent rewriting, we focus on the set described by Proposition 14. Since, now, the form of each view that can be used in a bag-set equivalent rewriting is a generalization of a duplicate-extension of a subexpression of Q's body, we can easily conclude that a similar subset of variables of the view is required in its head under bag-set semantics as well. The following proposition formally describes this result.

Proposition 16. *Let Q be a CQ and V be a view. Then there is a V-subgoal which covers S in a bag-set equivalent rewriting of Q using V,* ***if and only if*** *there is a substitution θ over V such that the following hold:*

1. *the body of the canonical representation of $\theta(V)$ is identical to the canonical representation of S,*

2. *for each variable X of V such that $\theta(X)$ is contained in $lvars(Q', S)$, X is a distinguished variable of V; where Q' is obtained from Q by*

adding a copy of each subgoal g of S which is also covered by another view-subgoal of R, if there exists such g; otherwise Q' is Q,

3. *for every two variables Y, Z of V such that $Y \neq Z$ and $\theta(Y) = \theta(Z)$, Y and Z are distinguished variables of V.*

Proof. This proof is similar to the proof of Proposition 14. □

Summarizing the results of this section we have that the form of each view V that can be used by a bag-set equivalent rewriting of a CQ Q is specified by Propositions 15 and 16. In addition, in each bag-set equivalent rewriting R of a CQ Q there is a subset of view-subgoals that covers all the subgoals of Q; and the residual subgoals provide only duplicate subgoals in the expansion of R. However, as it is proved in [ACGP07] (see also in Section 2.5) these residual view-subgoals are redundant (they neither give a more efficient rewriting nor are required for the existence of a rewriting).

4.2.2 Representative set of solutions for the bag-set-oriented view selection problem

Focusing on the bag-set-oriented view selection problem, the space of viewsets described in the previous section is infinite. In particular, the viewsets which can be constructed by considering duplicate-extensions of subexpressions of a CQ in a given workload are infinite (since the set of duplicate-extensions of an expression is infinite). However, it is proved in [ACGP07] that there is an optimal viewset for a given input (if there exists any solution) such that each view definition in this viewset has at most exponentially many subgoals than the definition of a query in the workload. On the other hand, if the queries in the workload do not have self-joins the number of subgoals of each such view is bounded by the number of subgoals of the longest query definition (i.e., the query with the most subgoals).

In this section, we give a search space for finding at least one optimal viewset for a given bag-set-oriented view selection problem input (considering that the CQs included in the workload may or may not contain self joins). This space is closely related to the representative set of solutions under bag semantics described in Section 4.1.3.

As we noticed in Example 38, we cannot restrict to the space of subexpressions of the queries in order to find an optimal solution. Similarly however, with bag-oriented version of the problem if any relation of a given schema occurs at most once in the query of the workload, then we can find an optimal solution focusing on subexpressions of queries.

Proposition 17. *Let $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ be a conjunctive bag-set-oriented view selection problem input such that every relation in $\mathcal{S}$ appears at most once in a body of some query in $\mathcal{Q}$. If there exists a solution for $\mathcal{P}$, then there exists an optimal solution $\Lambda = (\mathcal{V}, \mathcal{R})$ such that each view in $\mathcal{V}$ is defined as a subexpression of a query in $\mathcal{Q}$.*

Proof. Considering that there is a solution for $\mathcal{P}$, we have that there is an optimal solution $\Lambda_o = (\mathcal{V}_o, \mathcal{R}_o)$ for $\mathcal{P}$ such that (1) each view in $\mathcal{V}_o$ is a generalization of subexpression of a CQ in $\mathcal{Q}$ and its definition does not have self-joins, and (2) each rewriting in $\mathcal{R}_o$ does not have self-joins (the existence of such a optimal solution is implied by Section 2.5 [ACGP07]). Without loss of generality we consider that for each query in $\mathcal{Q}$ a unique optimal rewriting is included in $\mathcal{R}_o$.

We, now, construct a viewset $\mathcal{V}$ from $\mathcal{V}_o$ as follows. For every view $V_o \in \mathcal{V}_o$ for which there is a unique V_o-subgoal in a body of a rewriting in $\mathcal{R}_o$, we add into $\mathcal{V}$ the view V whose definition is the view-expansion of this view-subgoal. By Proposition 7, however, we have that each view in $\mathcal{V}$ is a subexpression of body of a query in $\mathcal{Q}$. Hence, each view in $\mathcal{V}$ is bag-set contained to a view in $\mathcal{V}_o$.

In addition, we construct the set of rewritings $\mathcal{R}$ as follows. For each rewriting R_o in $\mathcal{R}_o$ we replace each view-subgoal with the head of the definition of the corresponding view in $\mathcal{V}$. The rewriting obtained by this procedure are included in $\mathcal{R}$. By the construction of $\mathcal{V}$ and $\mathcal{R}$, we easily conclude that there is at least one bag-set equivalent rewriting of each query in $\mathcal{Q}$ using $\mathcal{V}$, and these rewritings are as efficient as the corresponding rewritings in $\mathcal{R}_o$.

Consequently, $\mathcal{V}$ is also admissible for $\mathcal{P}$; which implies that $\Lambda = (\mathcal{V}, \mathcal{R})$ is an optimal solution for $\mathcal{P}$. □

The question, now, that arises is how we can find an optimal solution when the CQs in the workload have self-joins. In particular, we want to find a space of viewsets that decides the existence of an optimal solution for a given bag-set oriented input that may contain queries with self-joins (i.e., if there is an admissible viewset then an optimal viewset is included in this space).

In order to find such a space we investigate whether there exists a space of viewsets in which each view definition has a bounded number of subgoals. This is achieved by separating two cases of views, similar to the cases in the bag-oriented context; the views that cover a unique subexpression of a query in the workload, and the views that cover multiple subexpressions of queries' bodies. In the first case, similar to bag semantics, the views whose bodies are subexpressions of bodies of queries suffice. Intuitively, it is not useful under bag-set semantics either to generalize a subexpression more than needed or to duplicate multiple subgoals of a query subexpression (in order to find views in optimal viewsets that cover a single subexpression).

On the other hand, focusing on the views that cover multiple subexpressions of queries' bodies, we noticed that the lggs of subexpressions do not suffice (see Example 38). In particular, in an optimal solution for a bag-set-oriented view selection input, a single view in the optimal viewset can cover multiple subexpressions of queries (using the corresponding bag-set equivalent rewritings) which may have different number of subgoals; instead of the bag-oriented context in which each view covers subexpressions with the same number of subgoals. In addition, considering a bag-set-oriented view selection input, the Proposition 15 implies that the body of a view which is included in an optimal viewset and covers multiple subexpressions of queries in the set of optimal rewritings, is a common generalization of duplicate-extensions of the subexpressions that it covers. We, now, extend the concept of lgg in order to capture these cases. More specifically, we introduce the concept of *d-lgg* of expressions as follows.

Definition 13. Let $\mathcal{N}$ be a set of relation names and $Exp = \{E_1, E_2, \ldots, E_n\}$ be a set of expressions such that for each $g \in \mathcal{N}$ there is at least one atom in every expression in Exp with relation name g, and each expression in Exp has atoms in $\mathcal{N}$. We consider the set of duplicate-extensions $Exp' = \{E'_1, E'_2, \ldots, E'_n\}$ of $E_1, E_2, \ldots, E_n$, respectively, such that for every relation name $g \in \mathcal{N}$ the number of g-atoms is the same in all expressions of Exp'. Then we say that an lgg E of the expressions in Exp' is an **d-lgg** of the expressions in Exp if E does not have duplicate atoms.

Example 40. Let the following six expressions.

$E_1 = \ e(X,X), e(X,Y), g(a,a), g(a,d).$

$E_2 = \ e(X,Y), e(Y,X), g(X,X), g(b,d).$

$E_3 = \ e(X,X), e(X,Y), g(a,a).$

$E_4 = \ e(A_1,A_2), e(A_2,A_3), g(A_4,A_4), g(A_5,d).$

$E_5 = \ e(B_1,B_2), e(B_2,B_3), g(B_4,B_4), g(B_5,B_6).$

$E_6 = \ e(B_1,B_2), e(B_2,B_3), g(B_4,B_4), g(B_5,B_6), g(B_5,B_6).$

Notice that there is common generalization for E_1 and E_2; instead of E_1, E_2 and E_3 that they do not have any. In addition, E_4 is an lgg of E_1 and E_2; from which it follows that E_4 is also a d-lgg of E_1 and E_2. Consider, now, the following duplicate-extension of E_3.

$$E'_3 = \ e(X,X), e(X,Y), g(a,a), g(a,a).$$

Using, now, E_3', we can easily verify that E_5 is a d-lgg of E_2 and E_3. However, constructing the duplicate-extensions E_3'' and E_2', where

$$E_3'' = e(X,X), e(X,Y), g(a,a), g(a,a), g(a,a),$$
$$E_2' = e(X,Y), e(Y,X), g(X,X), g(b,d), g(b,d),$$

we can easily notice that although E_6 is an lgg of E_2' and E_3'', E_6 is not a d-lgg of E_2 and E_3 (because E_6 has duplicate atoms). □

Intuitively, the d-lgg is an lgg of duplicate-extensions of a set of expressions with no redundant atoms (i.e., without duplicate atoms). Although a duplicate-extension of an expression may have arbitrary many atoms, the d-lggs of a set of expressions have bounded number of atoms. This is formally proved by the following proposition.

Proposition 18. *Let a set of expressions $Exp = \{E_1, E_2, \ldots, E_n\}$ such that there is at least one d-lgg of Exp. Also consider that for each $i = 1, \ldots, n$, the expression E_i of Exp has n_i atoms. Then for each d-lgg E of the expressions in Exp, the number of atoms of E is at most equal to $\Pi_{i=1}^{n}(n_i)$.*

Proof. We will prove the case in which the set *Exp* contains only two expression; the E_1 and E_2. For the general case (i.e., the set *Exp* contains n expressions) the proof is straightforwardly generalized. Moreover, without loss of generality we consider the case that all atoms in both expressions have the same relation name.

Consider a d-lgg E of E_1 and E_2 such that E contains $(n_1 \cdot n_2 + 1)$ atoms. By definition of d-lgg there are two expressions E_1' and E_2' which are duplicate-extensions of E_1 and E_2, respectively, such that E is an lgg of E_1' and E_2'. In addition, E does not have duplicate atoms. Hence, there are two substitutions θ_1 and θ_2 over E such that $\theta_1(E)$ is identical to E_1' and $\theta_2(E)$ is identical to E_2'. Thus, for each atom g of E there is a pair (g_1, g_2) where g_1 is an atom of E_1 and g_2 is an atom of E_2 such that $\theta_1(g) = g_1$ and $\theta_2(g) = g_2$. Notice now that there are $n_1 \cdot n_2$ such different pairs. Hence, since E contains $(n_1 \cdot n_2 + 1)$ atoms, there are two atoms s_1 and s_2 of E such that $\theta_1(s_1) = \theta_1(s_2) = g_1'$ and $\theta_2(s_1) = \theta_2(s_2) = g_2'$, where g_1' is an atom of E_1 and g_2' is an atom of E_2. Constructing, now, the expression E' from E such that we replace both s_1 and s_2 with the atom s, where s is obtained by applying the most general unifier on $\{s_1, s_2\}$, the following happens: E' is a common generalization of E_1' and E_2', and an instance of E. This is, however, a contradiction, since E is an lgg of E_1' and E_2'. Consequently, E contains at most $(n_1 \cdot n_2)$ atoms. □

Considering, now, a conjunctive bag-set-oriented view selection, we can solve the problem (i.e., we find an optimal solution whenever a solution exists) by searching viewsets which contain view definitions whose bodies are either subexpressions of queries' bodies or d-lggs of queries' bodies. This result is formally proved as follows.

Proposition 19. *Let $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ be a conjunctive bag-set-oriented view selection problem input. If there is any optimal solution for $\mathcal{P}$ then there is an optimal solution $\Lambda = (\mathcal{R}, \mathcal{V})$ for $\mathcal{P}$ such that the body of each view in $\mathcal{V}$ is either a subexpression of the body of a CQ in $\mathcal{Q}$ or a d-lgg of subexpressions of the bodies of CQs in $\mathcal{Q}$.*

Proof. Consider that there is an optimal solution $\Lambda_o = (\mathcal{R}_o, \mathcal{V}_o)$ for $\mathcal{P}$. Without lost of generality we suppose that there is no view in $\mathcal{V}'$ that isn't used by a rewriting in $\mathcal{R}_o$. We will prove the proposition by constructing an optimal solution $\Lambda = (\mathcal{R}, \mathcal{V})$ for $\mathcal{P}$ such that the body of each view in $\mathcal{V}$ is either a subexpression of the body of a query in $\mathcal{Q}$ or a d-lgg of subexpressions of (the bodies of) queries in $\mathcal{Q}$

We construct a viewset $\mathcal{V}$ from $\mathcal{V}_o$ as follows. For every view $V_o \in \mathcal{V}_o$ for which there is a single V_o-subgoal in a body of a unique rewriting in $\mathcal{R}_o$, we add into $\mathcal{V}$ the view V whose definition is the view-expansion of this view-subgoal. On the other hand, for each view $V_o \in \mathcal{V}_o$ for which multiple V_o-subgoals appear in rewritings in $\mathcal{R}_o$, we add into $\mathcal{V}$ the view V whose definition is the d-lgg of all view-expansions of the V_o-subgoals.

In addition, we construct the set of rewritings $\mathcal{R}$ as follows. For each rewriting R in $\mathcal{R}$ we replace each view-subgoal v with $\theta(v')$, where v' is the head of the definition of the corresponding view in $\mathcal{V}$ and θ is the appropriate substitution such that the bodies of the canonical representations of view-expansion of v and $\theta(v')$ are identical. By the construction of $\mathcal{V}$ and $\mathcal{R}$, we easily conclude that there is at least one bag-set equivalent rewriting of each query in $\mathcal{Q}$ using $\mathcal{V}$, and these rewritings are as efficient as the corresponding rewritings in $\mathcal{R}_o$ (which is implied by considering monotonic cost models).

In order, now, to prove that $\Lambda = (\mathcal{V}, \mathcal{R})$ is an optimal solution for $\mathcal{P}$, we have to prove that $\mathcal{V}$ is admissible; i.e., satisfies the storage constraint L. This can be easily proved by the construction of $\mathcal{V}$; since for each view V in $\mathcal{V}$ and the corresponding view V_o in $\mathcal{V}_o$ we have that V_o is a generalization of duplicate extension of V. Hence, there is a variables-onto containment mapping from V_o' to V'; which implies that $V_o \sqsubseteq_{bs} V$ (Proposition 2). Thus, the size of $\mathcal{V}$ is at most equal to the size of $\mathcal{V}_o$ (i.e., $size(\mathcal{V}(D)) \leq L$); namely, $\mathcal{V}$ is admissible. Consequently, by definition of optimal solution, Λ is an optimal solution for $\mathcal{P}$. $\square$

Example 41. Consider the bag-set view selection problem input $\mathcal{P}' = (\mathcal{S}, \mathcal{Q}, \mathcal{D}', L')$ where $\mathcal{D}' = \mathcal{D} \cup \{e(y_1, y_2, y_3, w_5, w_6, d, d),\ e(y'_1, y'_2, y'_3, w'_5, w'_6, d', d')\}$ and $\mathcal{S}$, $\mathcal{Q}$, $\mathcal{D}$ are defined in Example 38. Moreover, the storage limit L' is 4 tuples. As we noticed in Example 38 the viewset $\mathcal{V} = \{V\}$, is admissible for $\mathcal{P}$; instead of $\mathcal{V}_1 = \{V_1\}$ and $\mathcal{V}_2 = \{V_2\}$ which are not (we remind that $\mathcal{V}_1(\mathcal{D})$ and $\mathcal{V}_2(\mathcal{D})$ contain 5 and 4 tuples and $L = 3$ tuples). In addition, notice that the views V_1, V_2 and V indicate different d-lggs of the bodies of Q_1 and Q_2 (see also in Example 39). However, the viewsets $\mathcal{V}_1$, $\mathcal{V}$ are not admissible for $\mathcal{P}'$; since both $\mathcal{V}(\mathcal{D}')$ and $\mathcal{V}_1(\mathcal{D}')$ contain 5 tuples.

On the other hand, since $\mathcal{V}_2(\mathcal{D}')$ contains 4 tuples (i.e., $size(\mathcal{V}_2(\mathcal{D}')) = L'$) and $\mathcal{V}_2$ gives bag-set equivalent rewritings for both queries in $\mathcal{Q}$, we conclude that $\mathcal{V}_2$ is admissible for $\mathcal{P}'$.

Consider, now, the views V'_1 and V'_2 defined as subexpressions of bodies of Q_1 and Q_2, respectively (especially they are identical to Q_1 and Q_2). The definitions of these views are the following (notice that their definitions are identical to that of the corresponding queries).

$V'_1 : v'_1(A, B, C, D, E, F, G)$:- $e(A, A, B, C, C, D, D), e(A, B, B, E, F, D, G)$.

$V'_2 : v'_2(A, B, C, D, E, F, G)$:- $e(A, B, B, C, C, D, D), e(A, A, B, E, F, D, G)$.

It is easy to see that there are bag-set equivalent rewritings of Q_1 and Q_2 using $\mathcal{V}' = \{V'_1, V'_2\}$; these also minimize the evaluation cost (i.e., they are optimal rewritings). Moreover, notice that $\mathcal{V}'(\mathcal{D}')$ contains 4 tuples; i.e., $\mathcal{V}'(\mathcal{D}')$ does not violates the storage constraint L'. Hence, $\mathcal{V}'$ is an optimal viewset for $\mathcal{P}'$. □

Here, notice that considering an optimal viewset $\mathcal{V}$ for a bag-set-oriented input $\mathcal{P}$ whose views are defined as specified by Proposition 19, there may be viewsets which are obtained by generalizing the views of $\mathcal{V}$ and are also optimal viewsets for $\mathcal{P}$. However, the size of each such viewset $\mathcal{V}'$ is at least equal to the size of $\mathcal{V}$ and the corresponding rewritings of the queries using $\mathcal{V}'$ may be less efficient than the rewritings obtained by using views in $\mathcal{V}$. The following example describes how further generalizations of the views can reduce the efficiency of the rewritings.

Example 42. Consider the bag-set view selection problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, L)$ where $\mathcal{S}$ contains the binary relation e, $\mathcal{Q} = \{Q\}$ and $\mathcal{D}$ contains the tuples $e(a_i, a_{i+1})$, with $i = 1, \ldots, 5$. Moreover, $L = 60$ tuples and the definition of Q is the following.

$Q : q(X_1, X_6)$:- $e(X_1, X_2), e(X_2, X_3), e(X_3, X_4), e(X_4, X_5), e(X_5, X_6)$.

Consider, now, the admissible viewset $\mathcal{V}$ that contains the following views.

$V_1 : v_1(A_1, A_3)$:- $e(A_1, A_2), e(A_2, A_3)$.
$V_2 : v_2(A_1, A_2)$:- $e(A_1, A_2)$.

It is easy to verify that the following CQ is a bag-set equivalent rewriting of Q using $\mathcal{V}$.

$R : q(X_1, X_6)$:- $v_1(X_1, X_3), v_2(X_3, X_4), v_1(X_4, X_6)$.

Also, consider the viewset $\mathcal{V}' = \{V_1', V_2\}$ obtained from $\mathcal{V}$ by generalizing the definition of V_1. The views are defined as follows.

$V_1' : v_1'(A_1, B_1, B_2, A_3)$:- $e(A_1, B_1), e(B_2, A_3)$.
$V_2 : v_2(A_1, A_2)$:- $e(A_1, A_2)$.

We can easily verify that $\mathcal{V}'$ is also admissible for $\mathcal{P}$ since $size(\mathcal{V}'(\mathcal{D})) < L$ and there is a bag-set equivalent rewriting of Q using $\mathcal{V}'$. Such a rewriting is defined as follows.

$R' : q(X_1, X_6)$:- $v_1'(X_1, X_2, X_2, X_3), v_2(X_3, X_4), v_1'(X_4, X_3, X_3, X_6)$.

Notice, here, that R' is obtained from R by replacing each V_1-subgoal with a V_1'-subgoal. Since, now, V_1''s body is a generalization of V_1, evaluating R in $\mathcal{V}(\mathcal{D})$ is more efficient than evaluating R' in $\mathcal{V}'(\mathcal{D})$. To verify this notice that the evaluation of R' requires additional selections and projections (instead of R); since $V_1'(\mathcal{D})$ contains a superset of the information stored in $V_1(\mathcal{D})$. $\square$

Consider, now, bag-set-oriented view selection problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, L)$ and three subexpressions S, S_1 and S_2 of the bodies of queries Q, Q_1, Q_2 in $\mathcal{Q}$ such that there is a d-lgg of S_1 and S_2. We notice, here, that Q_1 and Q_2 may be the same query. However, although S_1 and S_2 may also be identical, they represent different subexpressions. Using Proposition 16, we construct a view V (called *subexpression view*) as follows: the body of V is identical to S and its head contains the $lvars(Q', S)$ where Q' is obtained by adding to Q a copy of each subgoal which is also covered by another view, if there exists such a subgoal; otherwise Q' is identical to Q. In addition, we can construct the view V' whose body is the d-lgg of S_1 and S_2 using the following steps:

1. We construct the subexpression views V_1 and V_2 using the subexpressions S_1 and S_2, respectively, as previously defined.

2. By considering V_1 and V_2 as queries we construct V' with body the d-lgg of the bodies of V_1 and V_2. Moreover, using Proposition 16, we put in the head of V' the minimum required set of variables such that treating both V_1 and V_2 as queries, V' covers both the bodies of V_1 and V_2. V' is said to be *d-lgview* of the views V_1 and V_2; and denoted as $dlgview(V_1, V_2)$.

An example of a d-lgview was given in Example 38; where V, V_1 and V_2 are d-lgviews. The construction of d-lgview can be easily generalized for more than two subexpression views.

The following theorem summarizes the results given by Proposition 16 and Proposition 19, and shows that if there is a solution for a given bag-set-oriented problem input then there is an optimal viewset that contains only subexpression views and d-lgviews.

Theorem 16. *Let a bag-set-oriented view selection input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$. If there is a solution for $\mathcal{P}$, then there exists an optimal solution $\Lambda = (\mathcal{V}, \mathcal{R})$ such that each view in $\mathcal{V}$ is either a subexpression view of a query in $\mathcal{Q}$ or a d-lgview of subexpression views of queries in $\mathcal{Q}$.*

The class of solutions constructed using the previous theorem is a *representative set of solutions* for the given bag-set-oriented view selection problem input $\mathcal{P}$. In addition, by easily modifying the LGG-VSB algorithm (Section 4.1.4) we solve the bag-set-oriented view selection problem. In particular, we replace the Step 1 of the LGG-VSB algorithm with the following step:
1. Let $\mathcal{V}$ be a set of viewsets constructed as follows: Each $\mathcal{V}' \in \mathcal{V}$ is of the form $\mathcal{V}' = \mathcal{V}_1 \cup \cdots \cup \mathcal{V}_n$, where n is the number of queries in $\mathcal{Q}$ and each viewset $\mathcal{V}_i$ is obtained from the query $Q_i \in \mathcal{Q}$ as follows:

- Let P_i be a minimal cover (i.e., the union of the blocks gives the body of Q_i and if we drop any block, we drop at least one distinct subgoal of Q_i) of the subgoals of Q_i.
- For each block $B_j \in P_i$, add a view definition $V_{i,j}$ in $\mathcal{V}_i$ whose body consists of the atoms in B_j and whose $vars(head(V_{i,j}))$ is the set $lvars(Q'_i, B_j)$; where Q'_i is obtained by adding in the body of Q_i a copy for each atom g of B_j which is also contained in a block B_k, with $k \neq j$.

Then, in Step 3 of the algorithm, we compute the d-lgview of views in $\mathcal{M}$ (instead of the $lgview(\mathcal{M})$) and we set the resulted view to V_l. Using these modifications the algorithm outputs the representative set of optimal solution for a given bag-set-oriented input.

4.3 Chain and Path queries

In this section, we further study the bag- and bag-set-oriented view selection problem through special cases analysis. In particular, we focus on the cases that the query workload is a set of either chain queries or path queries.

A *chain query* is a conjunctive query of the following form:

$$Q : q(X_0, X_n) \text{ :- } r_1(X_0, X_1), r_2(X_1, X_2), \ldots, r_n(X_{n-1}, X_n)$$

where $r_1, \ldots, r_n$, are binary relations and $X_0, X_1, \ldots, X_n$ are variables. If the relation symbols r_1, ..., r_n are identical then the query is called *path query* of length n, denoted as P_n.

The main results of this section are summarized as follows:

1. Subsection 4.3.1 demonstrates that either for a bag-oriented or for a bag-set-oriented problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$, where $\mathcal{Q}$ is a workload of chain queries, we cannot restrict the space of optimal solutions by searching admissible viewsets which contain only *chain-views*, i.e. views defined by chain queries.

2. Subsection 4.3.2 demonstrates that for a problem input $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$, where $\mathcal{Q}$ is a workload of path queries, if there exists a solution for $\mathcal{P}$, then there is at least one optimal solution for $\mathcal{P}$ which is constructed by an admissible viewset containing only path views (Theorem 17). However, in the case of bag-set semantics the path-viewsets do not suffice (Proposition 23).

4.3.1 Chain-query workload

In this section we study the view selection problem for workloads containing only chain-queries. In particular, we focus our attention on whether there is an optimal solution constructed by a set of chain-views. Unfortunately, as the following proposition shows, there are cases in which none of the optimal solutions is constructed by a set of chain-views.

Proposition 20. *There exists at least one bag-oriented view selection problem input $\mathcal{P} = (S, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ such that:*

- *$\mathcal{Q}$ is a set of chain queries, and*
- *$\mathcal{P}$ has optimal solutions but there is **no** optimal solution $\Lambda = (\mathcal{V}, R)$ such that $\mathcal{V}$ contains only chain queries.*

Proof. The following example proves this proposition. □

Example 43. Consider a query workload $\mathcal{Q} = \{Q\}$ on a database schema $\mathcal{S}$ that contains the binary relations r_1, r_2 and r_3, where Q is the following chain query:

$$Q : q(X, Y) \text{ :- } r_1(X, Z), r_2(Z, W), r_3(W, Y).$$

Consider also the following five viewsets $\mathcal{V}_i$, $i \in \{1, 2, 3, 4, 5\}$:

$\mathcal{V}_1 = \{V_{11}, V_{12}\}$, where:

$$V_{11} : v_{11}(X, Z, W, Y) \text{ :- } r_1(X, Z), r_3(W, Y).$$
$$V_{12} : v_{12}(X, Y) \text{ :- } r_2(X, Y).$$

$\mathcal{V}_2 = \{V_{21}, V_{22}\}$, where:

$$V_{21} : v_{21}(X, Y) \text{ :- } r_1(X, Z), r_2(Z, Y).$$
$$V_{22} : v_{22}(X, Y) \text{ :- } r_3(X, Y).$$

$\mathcal{V}_3 = \{V_{31}, V_{32}\}$, where:

$$V_{31} : v_{31}(X, Y) \text{ :- } r_2(X, Z), r_3(Z, Y).$$
$$V_{32} : v_{32}(X, Y) \text{ :- } r_1(X, Y).$$

$\mathcal{V}_4 = \{V_{41}\}$, where:

$$V_{41} : v_{41}(X, Y) \text{ :- } r_1(X, Z), r_2(Z, W), r_3(W, Y).$$

$\mathcal{V}_5 = \{V_{51}, V_{52}, V_{53}\}$, where:

$$V_{51} : v_{51}(X, Y) \text{ :- } r_1(X, Y).$$
$$V_{52} : v_{52}(X, Y) \text{ :- } r_2(X, Y).$$
$$V_{53} : v_{53}(X, Y) \text{ :- } r_3(X, Y).$$

Observe that the above viewsets are all possible viewsets constructed as described in Section 4.1.

Suppose a bag-valued database instance $\mathcal{D} = \{(r_1$(a,b);5), (r_2(b,c);10), (r_3(c,d);5)$\}$. Considering a storage limit L=35 tuples, the following viewsets:

$$\mathcal{V}_1(\mathcal{D}) = \{(v_{11}(a, b, c, d); 25), (v_{12}(b, c); 10)\}$$
$$\mathcal{V}_5(\mathcal{D}) = \{(v_{51}(a, b); 5), (v_{52}(b, c); 10), (v_{53}(c, d); 5)\}$$

do not violate the storage limit constraint. In contrast, the viewsets:

$$\mathcal{V}_2(\mathcal{D}) = \{(v_{21}(a, c); 50), (v_{22}(c, d); 5)\}$$
$$\mathcal{V}_3(\mathcal{D}) = \{(v_{31}(a, c); 50), (v_{32}(c, d); 5)\}$$
$$\mathcal{V}_4(\mathcal{D}) = \{(v_{41}(a, c); 250)\}$$

do violate it. Thus, $\Lambda = (\mathcal{V}_1, R)$ and $\Lambda' = (\mathcal{V}_5, R')$ are solutions for input $\mathcal{P}$, where the rewritings R and R' are the following:

$$R : q(X, Y) \text{ :- } v_{11}(X, Z, W, Y), v_{12}(Z, W).$$
$$R' : q(X, Y) \text{ :- } v_{51}(X, Z), v_{52}(Z, W), v_{53}(W, Y).$$

Using the cost model presented in Section 2.5, the costs of Λ and Λ' are $C(R, \mathcal{V}_1(\mathcal{D})) = 55$ and $C(R', \mathcal{V}_4(\mathcal{D})) = 325$ respectively. As a consequence, Λ is an optimal solution for the bag-oriented view selection problem input $\mathcal{P}$. □

In the case of bag-set view selection problem, the same result holds. The following proposition shows this result.

Proposition 21. *There exists at least one bag-set-oriented view selection problem input* $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ *such that:*

- $\mathcal{Q}$ *is a set of chain queries, and*

- *$\mathcal{P}$ has optimal solutions but there is **no** optimal solution $\Lambda = (\mathcal{V}, R)$ such that $\mathcal{V}$ contains only chain queries.*

Proof. The following example proves this proposition. $\square$

Example 44. Let a query workload $\mathcal{Q} = \{Q\}$ on a database schema $\mathcal{S}$ that contains the binary relations r_1, r_2 and r_3, where Q is the following chain query:

$$Q : q(X,Y) \text{ :- } r_1(X,Z), r_2(Z,W), r_1(W,Y).$$

Consider also the following seven views:

$$V_1 : v_1(X,Y) \text{ :- } r_1(X,Y).$$
$$V_2 : v_2(X,Y) \text{ :- } r_2(X,Y).$$
$$V_3 : v_3 \text{ :- } r_1(X,Z), r_2(Z,W), r_1(W,Y).$$
$$V_{12} : v_{12}(X,Y) \text{ :- } r_1(X,Z), r_2(Z,Y).$$
$$V_{21} : v_{21}(X,Y) \text{ :- } r_2(X,Z), r_1(Z,Y).$$
$$V_{41} : v_{11}(X,Z,W,Y) \text{ :- } r_1(X,Z), r_1(W,Y).$$
$$V_{42} : v_4(X,Z,W,Y) \text{ :- } r_1(X,Z), r_2(W,Y).$$

Notice that the first five views are defined by chain queries whose body is a subexpression of Q. In addition, let a database instance $\mathcal{D}$ of $\mathcal{S}$ which contains the tuples $r_1(a,a)$, $r_1(a,b)$, $r_1(b,a)$ and $r_2(a,a)$. Considering, now, the viewsets $\mathcal{V}_1 = \{V_1, V_2\}$, $\mathcal{V}_2 = \{V_3\}$, $\mathcal{V}_3 = \{V_1, V_{12}\}$ and $\mathcal{V}_4 = \{V_1, V_{21}\}$, we can easily verify that these are the only viewsets that contain chain views (i.e., views whose definitions are given by a chain query), give a bag-set equivalent rewriting for Q and do not have any redundant view (i.e., we cannot drop any view and the produced viewset also gives bag-set equivalent rewriting for Q). Moreover, supposing a storage limit $L = 3$ tuples, we conclude that none of the above viewsets is admissible for $\mathcal{P} = (\mathcal{S}, \mathcal{Q}, \mathcal{D}, L)$ (because both $\mathcal{V}_1(\mathcal{D})$ and $\mathcal{V}_2(\mathcal{D})$ contain 4 tuples, and both $\mathcal{V}_3(\mathcal{D})$ and $\mathcal{V}_4(\mathcal{D})$ contain 5 tuples).

On the other hand, let the viewsets $\mathcal{V}_5 = \{V_{41}\}$ and $\mathcal{V}_6 = \{V_{42}\}$. Notice that none of them contain chain views. Moreover, the viewset $\mathcal{V}_5$ contains a subexpression view of Q and give a bag-set equivalent rewriting. This viewset is not admissible for $\mathcal{P}$ (because $\mathcal{V}_5(\mathcal{D})$ contain 9 tuples), as well. The viewset $\mathcal{V}_6$, however, is admissible for $\mathcal{P}$ (we can easily verify that this viewset is optimal for $\mathcal{P}$), since $\mathcal{V}_6(\mathcal{D})$ contains 3 tuples and gives a bag-set equivalent rewriting of Q. The definition of such a rewriting is given as follows.

$$R : q(X,Y) \text{ :- } v_{42}(X,Z,Z,W), v_{42}(W,Y,Z,W).$$

$\square$

4.3.2 Path-query workload

In this section we study the view selection problem for *path-query workloads* (i.e. workloads of path queries). Unlike to the problem for chain query workloads in which we cannot reduce the search space to the class of chain views, for path-query workloads under bag semantics we can reduce the search space even more. The main result of this section, presented by the following theorem, is that whenever the workload is a set of path-queries, we can focus on *path-viewsets* whose views have at most as many subgoals as the length of the longest path-query in the workload.

Theorem 17. *Let $\mathcal{P} = (S, \mathcal{Q}, \mathcal{D}, \mathcal{L})$, be a conjunctive bag-oriented view selection input, and $\mathcal{Q}$ contains a set of path queries. If there exists a solution for $\mathcal{P}$, then there is an optimal solution $\Lambda = (\mathcal{V}, \mathcal{R})$ for $\mathcal{P}$ such that:*

1. *each view in $\mathcal{V}$ is defined as a path of the same relation as a query $Q \in \mathcal{Q}$,*
2. *every view in $\mathcal{V}$ has at most n subgoals, where n is the length of the longest query in $\mathcal{Q}$,*
3. *every $R \in \mathcal{R}$ is a chain query.*

Proof. Consider that there is a solution for $\mathcal{P}$. Hence, Theorem 15 implies that there exists an optimal solution $\Lambda = (\mathcal{V}_o, \mathcal{R}_o)$ for $\mathcal{P}$ such that each view in $\mathcal{V}_o$ is either a subexpression view or an lgview of subexpression views of queries in $\mathcal{Q}$. We will construct a set $\mathcal{V}$ of path-views from $\mathcal{V}_o$ such that each view in $\mathcal{V}$ is bag contained in a view of $\mathcal{V}_o$. Moreover, the views are constructed in such a way that optimal rewritings of queries in $\mathcal{Q}$ using $\mathcal{V}$ exist. Moreoever, without loss of generality we consider that all path queries in $\mathcal{Q}$ are referred to the same binary relation e of $\mathcal{S}$.

We construct, now, the viewset $\mathcal{V}$ from $\mathcal{V}_o$ as follows: for every view V_o in $\mathcal{V}_o$ there is a path-view V in $\mathcal{V}$ such that V is a path-view of length n, where n is the number of subgoals of V_o. In addition, V is also referred to relation e. The size, now, of V_o is either greater than or equal to V. In order to verify this, we have that there is a substitution θ over the body of V_o such that $\theta(body(V_o))$ is identical to a subexpression S of a body of a path query Q in $\mathcal{Q}$. By definition of path query we conclude that the form of S is the following:

$$S = e(X_1, X_2), e(X_3, X_4), \ldots, e(X_{2n-1}, X_{2n})$$

where for every $k = 1, \ldots, n$, we have that $X_{2k-1} \neq X_{2k}$ and the variables X_{2k} and X_{2k+1} are not necessarily different. In addition the definition of V

is the following:

$$V : v(Y_0, Y_n) \text{ :- } e(Y_0, Y_1), e(Y_1, Y_2), \ldots, e(Y_{n-1}, Y_n)$$

Hence, considering the substitution θ' over S such that $\theta'(X_1) = Y_0$, $\theta'(X_{2n}) = Y_n$ and for each $k = 1, \ldots, (n-1)$, $\theta'(X_{2k}) = \theta'(X_{2k+1}) = Y_{n-1}$, we conclude that $\theta'(S) = body(V)$. Thus, $\theta'(\theta(body(V_o))) = body(V)$. We refer to the substitution over V_o obtained by the composition of θ and θ' as ϕ (i.e., $\phi(body(V_o)) = body(V)$). Consequently, using the following proposition we have that $size(V_0(\mathcal{D})) \geq size(V(\mathcal{D}))$; hence, $size(\mathcal{V}_0(\mathcal{D})) \geq size(\mathcal{V}(\mathcal{D}))$.

Proposition 22. *Let two CQs Q_1, Q_2 defined over the same schema $\mathcal{S}$ and a substitution θ over Q_1 such that $\theta(body(Q_1)) = body(Q_2)$ and $vars(head(Q_2)) \subseteq vars(\theta(head(Q_1)))$. Then for every bag-valued database instance $\mathcal{D}$ of $\mathcal{S}$ we have that $size(Q_1(\mathcal{D})) \geq size(Q_2(\mathcal{D}))$.*

Proof. Consider an arbitrary bag-valued database $\mathcal{D}$ of $\mathcal{S}$. In addition, consider the CQ Q'_1 obtained from Q_1 such that $body(Q'_1) = body(Q_1)$ and $\theta(head(Q'_1)) = head(Q_2)$. Since, now, $vars(head(Q_2)) \subseteq vars(\theta(head(Q_1)))$ we have that $Q'_1(\mathcal{D})$ is obtained by applying a bag-projection on $Q_1(\mathcal{D})$. Hence, by definition of bag projection we have that $size(Q_1(\mathcal{D})) \geq size(Q'_1(\mathcal{D}))$. In addition, by construction of Q'_1 we have that $Q_2 \sqsubseteq_b Q'_1$; hence $size(Q'_1(\mathcal{D})) \geq size(Q_2(\mathcal{D}))$. Thus, $size(Q_1(\mathcal{D})) \geq size(Q_2(\mathcal{D}))$. □

In order to show, now, that $\mathcal{V}$ is optimal for $\mathcal{P}$ we have to find a set of optimal rewritings $\mathcal{R}$ using $\mathcal{V}$; one for each CQ Q in $\mathcal{Q}$. The rewritings $\mathcal{R}$ are obtained from $\mathcal{R}_o$ as follows. For each CQ Q in $\mathcal{Q}$ and its rewriting R_o in $\mathcal{R}_o$ we have that there is an optimal query plan (it gives the less evaluation cost) of R_o which has the following form:

$$(\ldots((g_1 \bowtie g_2) \bowtie g_3) \bowtie \ldots \bowtie g_m),$$

where $g_1, \ldots, g_m$ are the view-subgoals of R_o. Replacing, now, each view-subgoal g_i with the head of corresponding view in $\mathcal{V}$ we add in $\mathcal{R}$ the chain query R with the following definition:

$$R : q(X_0, X_n) \text{ :- } v_1(X_0, X_1), v_2(X_1, X_2), \ldots, v_m(X_{m-1}, X_m)$$

where $head(R) = head(Q)$ and for each $i = 1, \ldots, m$ we have that V_i is the view in $\mathcal{V}$ which was constructed by the definition of the view-subgoal g_i. By the construction of $\mathcal{V}$, we easily conclude that R is a bag equivalent rewriting of Q using $\mathcal{V}$; which implies that $\mathcal{V}$ is admissible for $\mathcal{P}$. In addition, considering the following query plan for R,

$$(((v_1 \bowtie v_2) \bowtie v_3) \bowtie \ldots \bowtie v_m),$$

we can conclude that $C(R, \mathcal{V}(\mathcal{D})) = C(R_o, \mathcal{V}_o(\mathcal{D}))$. This can be verified using Proposition 22 and the assumption that the cost model we consider is monotonic. In particular, for each $i = 1, \ldots, m$, the intermediate relation IR_i of the above optimal query plan for R_o, which is obtained by joining the relations $g_1, \ldots, g_i$, can be written as a CQ with the corresponding view-subgoals of R_o in its body (i.e., the view-subgoals $g_1, \ldots, g_i$ are placed in its body). Constructing, now, the expansion of the definition of IR_i we get a CQ whose body is defined as subexpression of Q (since the body of the view-expansion of each view-subgoal of a bag equivalent rewriting is a subexpression of the corresponding CQ, up to a renaming substitution). Similarly, for each $i = 1, \ldots, m$, the intermediate relation IR'_i of the above query plan for R can be also defined as CQ (the body of IR'_i has in its body the subgoals $v_1, \ldots, v_i$). Hence, by construction of the definitions of IR_i and IR'_i, we have that there is a substitution h over the expansion of IR_i such that $h(body(IR_i^{exp})) = body(IR_i'^{exp})$ and $vars(head(IR_i)) \subseteq vars(h(head(IR'_i)))$. Thus, Proposition 22 implies that $size(IR_i) \geq size(IR'_i)$. Consequently, since we consider monotonic cost models we have that $\mathcal{V}$ is an optimal viewset for $\mathcal{P}$ and $\Lambda = (\mathcal{V}, \mathcal{R})$ is an optimal solution for $\mathcal{P}$. □

Consequently, we may restrict our attention in searching optimal solutions constructed by path-viewsets. In this case, the number of admissible viewsets is exponential to the number of subgoals of the path-queries in the workload. This exponential bound is implied by the reduction of the problem of searching path-viewsets to the integer-partitioning problem [AE04].

Example 45. Consider a query workload $\mathcal{Q} = \{P_3\}$, where P_3 is the following path-query of length 3:

$$P_3 : q(X, Y) \text{ :- } edge(X, Z), edge(Z, W), edge(W, Y)$$

Consider also the following three viewsets $(\mathcal{V}_i)$, $i \in \{1, 2, 3\}$:

$\mathcal{V}_1 = \{V_3\}$, where:

$$V_3 : v_3(X, Y) \text{ :- } edge(X, Z), edge(Z, W), edge(W, Y)$$

$\mathcal{V}_2 = \{V_2, V_1\}$, where:

$$V_2 : v_2(X, Y) \text{ :- } edge(X, Z), edge(Z, Y)$$
$$V_1 : v_1(X, Y) \text{ :- } edge(X, Y)$$

$\mathcal{V}_3 = \{V_1\}$, where:

$$V_1 : v_1(X, Y) \text{ :- } edge(X, Y)$$

Notice that the above path-viewsets form all the path-partitions (i.e., each block is the body of a path-query) of the body of P_3. In addition, each of them gives a set of bag equivalent rewritings of P_3. Notice also that every other path-viewset, which can be used to rewrite P_3, gives one of the $\mathcal{V}_i$, $i \in \{1, 2, 3\}$, by eliminating at least one path-view. For example, the viewset $\mathcal{V}_4$ that contains the path-views V_1 and V_3 can be used to rewrite

P_3; and eliminating one of the two views we obtain either $\mathcal{V}_1$ or $\mathcal{V}_3$. The equivalent rewritings $\mathcal{R}_i$ of P_3 using $\mathcal{V}_i$ respectively, $i \in \{1, 2, 3\}$, are the following:

$\mathcal{R}_1 = \{R_1\}$, where:

$$R_1 : q(X, Y) \text{ :- } v_3(X, Y)$$

$\mathcal{R}_2 = \{R_{21}, R_{22}\}$, where:

$$R_{21} : q(X, Y) \text{ :- } v_1(X, Z), v_2(Z, Y)$$
$$R_{22} : q(X, Y) \text{ :- } v_2(X, Z), v_1(Z, Y)$$

$\mathcal{R}_3 = \{R_3\}$, where:

$$R_3 : q(X, Y) \text{ :- } v_1(X, Y), v_1(X, Y), v_1(X, Y)$$

Moreover, observe that the above path-viewsets are related to the partitions of the length of P_3. In particular, the partitions of 3 are the following:

$$\{3\}$$
$$\{2, 1\}$$
$$\{1, 1, 1\}$$

Eliminating, now, duplicate numbers from the partitions of 3, we get the above path-viewsets as sets of lengths of path-views. □

Based on Theorem 17, we can improve the LGG-VSB for workloads containing only path-queries. In particular, when we know that the workload $\mathcal{Q}$ consists of n path-queries of the same relation, steps 1-4 of LGG-VSB can be replaced by the following step:

- Each $\mathcal{V}_{\mathcal{I}} \in \mathcal{V}$ contains a path-view V_k of length k, for every distinct integer $k \in \mathcal{I}$, where the set of integers $\mathcal{I}$ is of the form $\mathcal{I} = \mathcal{I}_{k_1} \cup \cdots \cup \mathcal{I}_{k_n}$, and $\mathcal{I}_{k_i}$ is a partition of the length of path-query $P_{k_i} \in \mathcal{Q}$, $i \in \{1, \ldots n\}$; the partitions of an integer can be computed using an algorithm from [ZS98].

Focusing, now, on the bag-set-oriented version of this problem (i.e., the workload contains path queries and we consider bag-set semantics) the path-viewsets do not suffice in order to find an optimal solution. This, result is formally given by the following proposition.

Proposition 23. *There exists at least one bag-set-oriented view selection problem input $\mathcal{P} = (S, \mathcal{Q}, \mathcal{D}, \mathcal{L})$ such that:*

- *$\mathcal{Q}$ is a set of path queries, and*
- *$\mathcal{P}$ has optimal solutions but there is **no** optimal solution $\Lambda = (\mathcal{V}, R)$ such that $\mathcal{V}$ contains only path queries.*

Proof. The following example proves this proposition. □

Example 46. Consider a path-query workload $\mathcal{Q} = \{P_2, P_3\}$ over the schema $\mathcal{S}$ that contains the binary relation *edge*, where P_2 and P_3 are defined as follows.

$P_2 : p_2(X,Y)$:- $edge(X,Z), edge(Z,Y)$.
$P_3 : p_3(X,Y)$:- $edge(X,Z), edge(Z,W), edge(W,Y)$

The path-viewsets that do not have redundant views and give bag-set equivalent rewritings of the queries in $\mathcal{Q}$ are the following:
$\mathcal{V}_1 = \{V_2, V_3\}$, where:
$V_2 : v_2(X,Y)$:- $edge(X,Z), edge(Z,Y)$.
$V_3 : v_3(X,Y)$:- $edge(X,Z), edge(Z,W), edge(W,Y)$
$\mathcal{V}_2 = \{V_2, V_1\}$, where:
$V_2 : v_2(X,Y)$:- $edge(X,Z), edge(Z,Y)$
$V_1 : v_1(X,Y)$:- $edge(X,Y)$
$\mathcal{V}_3 = \{V_1\}$, where:
$V_1 : v_1(X,Y)$:- $edge(X,Y)$

Supposing, now, a storage limit $L = 2$ tuples and the database instance $\mathcal{D} = \{edge(a,b), edge(b,c), edge(c,d)\}$ of $\mathcal{S}$, we conclude that none of the above path-viewsets is admissible ($\mathcal{V}_1(\mathcal{D})$ and $\mathcal{V}_3(\mathcal{D})$ contain 3 tuples, and $\mathcal{V}_2(\mathcal{D})$ contains 5 tuples). Considering, however, the viewset $\mathcal{V} = \{V\}$, where the definition of V is the following,

$V : v(A,B,C)$:- $edge(A,B), edge(B,C)$

we have that $\mathcal{V}(\mathcal{D})$ contains the tuples $v(a,b,c)$ and $v(b,c,d)$; hence, $\mathcal{V}$ satisfies the storage constraint L. In addition, the view V, which is not a path-view, gives the bag-set equivalent rewritings R_2 and R_3 of P_2 and P_3, respectively. The definitions of these rewritings are the following.

$R_2 : q(X,Y)$:- $v(X,Z,Y)$
$R_3 : q(X,Y)$:- $v(X,Z,W), v(Z,W,Y)$

Consequently, although there is a solution (i.e., there is an optimal solution) for the bag-set-oriented input $\mathcal{P} = (S, \mathcal{Q}, \mathcal{D}, \mathcal{L})$, there is no admissible path-viewset for $\mathcal{P}$.

□

4.4 Conclusions and Future Work

In this chapter, we have studied the problem of view selection under bag and bag-set semantics. In particular, for both versions of the problem, we aimed to limit the search space of candidate views, given a workload of CQs. We studied the problem in a rigorous way, and found sound and complete algorithms to select views for a query workload. Thus we improved previous results by exploiting very refined characterizations of views that participate in equivalent rewritings. Then we studied the problem in special cases and showed that, in these cases, we can limit the search space even further, until we identified a case (path queries) where considering subexpressions (actually even we showed that not all subexpressions are necessary to be considered) of queries are proven to be sufficient. Moreover, we describe a technique of equivalently rewriting a CQ under bag-set semantics by giving the requirements that the views have to meet in order to be used by a bag-set equivalent rewriting of a certain CQ.

There is a lot to be done for future work. For instance, under which restrictions do subexpressions suffice? Namely, an interesting problem to study is more special cases where considering subexpressions give good results either in rigorous sound and complete algorithms or in heuristics. Studying the exact complexity of the problem and especially finding tractable special cases is open. Many issues of the view selection problem remain open such as it is demonstrated in the discussion in Section 2.5.2. In addition, more constraints, such as minimal view maintenance cost, can be incorporated in the definition of the view selection problem. In this case, as an additional criteria is regarded, we expect a jump of the complexity. However, the space of optimal solutions may be refined.

Chapter 5

Heterogeneous Data Sources (XML data)

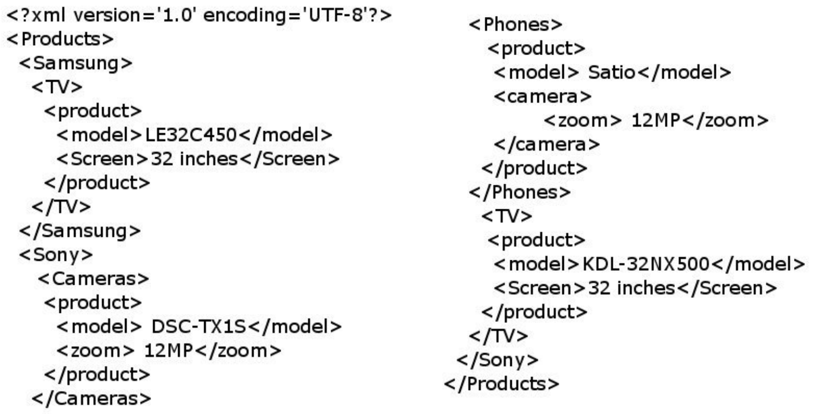
Nowadays, the Web is the most widespread and easily accessible information resource. This huge collection of information consists of arbitrary many data types; where most of them provide display information (e.g., HTML files). Most people see such Web data as Web documents, but these documents, rather than being manually composed, are increasingly generated automatically from conventional database systems (e.g., object-oriented database and relational database - see Chapter 2).

The conventional structured data models are usually described by a schema (e.g. relational data model, object-oriented data model). The schema is a collection of signatures which describes the structure of the data. This structural information implies efficient implementations of the data models. The efficiency, now, comes from the fact that the data must fit the schema, and the schema is known, in advance; hence data acquire real-life meaning.

In the Web, data are "schemaless". However, there is a way to conjecture a structure from the data; due to the way that they are constructed. The documents therefore have some regularity or some underlying meaningful structure which may or may not be understood by the user. In the current version of the Web, however, most of the major sites exploit this underlying structure in many ways (information integration, dynamic pages, better search capability etc.). This is usually achieved by providing most of their information using semi-structured documents. The preeminent type of this data model is the semistructured data; with most important manifestation of this idea: XML.

Extensible Markup Language (XML): The XML is a tag-based notation designed originally for "marking" documents, much like familiar HTML. However, while the tags in HTML provide display information, in XML the tags give semantic meaning. XML documents basically contain markup *elements* that have an open tag and an closed tag; where an open tag is text surrounded by triangular brackets, i.e., $< \cdots >$, and the corresponding closed tag is the same word with a slash at the beginning; and is also surrounded by triangular brackets.

A well-formed XML document must have exactly one top-level XML element (called root element) and all elements appearing in it must be properly nested (i.e., if an a-element is opened inside a b-element, then it must be closed inside the b-element). The names of the tags are completely defined by the user (XML allows the user to invent his own tags). Moreover, the nested structure of the elements implies that each element is described by its name (i.e., the name of the surrounded tags) and by its content (given by its subelements). An example of an XML document is illustrated in Figure 5.1. This document stores information about electronic products.



```
<?xml version='1.0' encoding='UTF-8'?>
<Products>
 <Samsung>
  <TV>
   <product>
    <model>LE32C450</model>
    <Screen>32 inches</Screen>
   </product>
  </TV>
 </Samsung>
 <Sony>
   <Cameras>
   <product>
    <model> DSC-TX1S</model>
    <zoom> 12MP</zoom>
   </product>
  </Cameras>
  <Phones>
   <product>
    <model> Satio</model>
    <camera>
        <zoom> 12MP</zoom>
    </camera>
   </product>
  </Phones>
   <TV>
   <product>
    <model>KDL-32NX500</model>
    <Screen>32 inches</Screen>
   </product>
  </TV>
 </Sony>
</Products>
```

Figure 5.1: Electronic Products

The document illustrated in Figure 5.1 has a root element named *Products*. The products, now, are firstly categorized according to their makes. Hence, two elements named by *Samsung* and *Sony* represent this grouping. In addition, for each of the above makes, the products are grouped into the following categories: *TV*, *Phones* and *Cameras*. Each product is represented by an element named by *product* and its characteristics are given by further elements and multiple string-values.

We can easily see that using this representation, the user provides real-life meaning to different pieces of information. Moreover, since the elements are completely nested, each XML document can be described by a tree-like representation. In this representation, an element-name is placed in each node and its children represent its subelements. Hence, a subtree of such a tree intuitivelly describes an element (i.e., the root of the subtree describes the name of the surrounded tags and its children describe its content) appearing in the corresponding document. Figure 5.2 describes the tree representation of the document illustrated in Figure 5.1. Therefore, we can notice that the tree in Figure 5.2 is rooted by the root element of the document (i.e., *Products*) and the parrent-children relationship describes the way the elements are nested in the document.

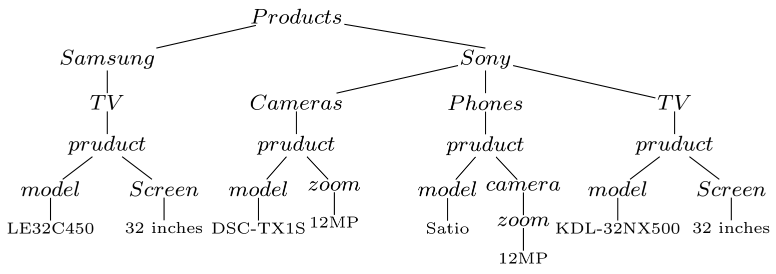


Figure 5.2: Tree representation of XML document

The XML documents can also be enforced by a "schema". Such a "schema" describes which elements appear in a document and the way these elements are nested inside the document. The documents that satisfy a given "schema" are called *valid*. A simple grammar used for the construction of such "schemas" is DTD (Document Type Definition). In addition, a more expressive grammar is given by XML Schema. In the following, we focus on well-formed XML documents which are not conformed to any "schema". Moreover, we describe each XML document using its tree-like representation.

Managing XML data: The management of the information appearing in a collection of XML data (the XML data are clustered into documents) is achieved by using XML administrative languages such as XPath, XSLT and XQuery [GMUW08]. Here, we focus on the simplest and most primitive of these languages, the XPath. The XPath constitutes the basis for most of the other XML administrative languages.

XPath is a language based on path expressions that allows the selection of parts of a given XML document. In addition it allows some minor computations resulting in values such as strings, numbers or booleans. The semantics of the language is based on a representation of the information content of an XML document as an ordered tree. An XPath expression consist usually of a series of steps that each navigate through this tree in a certain direction and select the nodes in that direction that satisfy certain properties.

Evaluating an XPath expression over an XML document, a completely ordering is regarded over the elements of the document. This ordering is implied by the top-down order of the elements appearing in the document. Using, now, this ordering elements that are semantically similar are considered distinct. For example, while the TV products of the document illustrated in Figure 5.1 have the same subelement *Screen*, the existence of the ordering implies that the *Screen*-elements are distinct. Hence, an expression guarantees that in its result every distinct element of the document appears once, but does not guarantees the elimination of semantically similar elements in its result. The elimination of similar elements can be achieved by using built-in functions; such as distinct-values which is applied on primitive types.

For example, consider the simple XPath expression $*//product/Screen$. This expression searches for all elements named by *Screen* which are subelements of an element *product*. The "star"-symbol at the beginning of the expression indicates that the expression can be applied on any document whose root is labeled by anything. The double slash means that we can search as deep as we want in order to find *product*-elements. The single slash between the element *product* and the element *Screen* indicates that each resulted element named by *Screen* must be a proper subelement of a *product*-element. Applying, now, the above expression on the XML document illustrated in Figure 5.1, we firstly search the subelements of the *Samsung*-element. Hence, since the element "$< Screen > 32\ inches < \backslash Screen >$" satisfies this condition, it is included in the result. Similarly, the element "$< Screen > 32\ inches < \backslash Screen >$" which appears inside the *Sony*-element, also satisfies the conditions. Hence, the element "$< Screen > 32\ inches < \backslash Screen >$" appears two times in the result of the expression.

In the following, we focus on the most important fragment of XPath expressions (named as patterns) which can also be represented by a tree-like form. Details about this fragment and the evaluation of such expressions are given in the following sections.

An overview of this chapter: In Section 5.1, we give the preliminaries of the XML trees and patterns. The semantics of the patterns are defined in Section 5.1.1. Focusing on further describing how patterns in a specific fragment search for elements in an XML tree, in Section 5.1.2, we define the core patterns. The problems of containment and equivalence of patterns are analyzed in Section 5.2. Finally, in the last section we describe the problem of answering patterns using XML views (Section 5.2.5).

5.1 XML trees and Patterns

A *directed path* (*path*, for short) p is a sequence $n_1, n_2, \ldots, n_k$ of nodes such that for each $i = 1, \ldots, k-1$ there is an edge (n_i, n_{i+1}). The number $(k-1)$ is called *length* of p, and represents the number of edges (n_i, n_{i+1}) appearing on p. The node n_1 is called *start node* of p and the node n_k is called *end node* of p. In addition, a *rooted tree* t is a directed graph with a designated node, denoted by root(t), such that every other node of t is reachable from $root(t)$ through a unique path (this path is referred as the *reachable path* of the certain node). In a labeled tree, every node n has a label which is given by the function *label* from the nodes of tree to a set of labels Σ; we write $label(n)$ to denote the label of n. We use $\mathcal{N}(t)$ and $\mathcal{E}(t)$ to denote the set of nodes and edges respectively, of a tree t.

Consider a rooted, labeled tree t and an edge $(n_1, n_2) \in \mathcal{E}(t)$. The Node n_1 is the *parent* of n_2, while n_2 is a *child* of n_1. Moreover, a node n_1 is an *ancestor* of n_2 (and n_2 is a *descendant* of n_1) if t has a path from n_1 to n_2 (i.e. the start node of p is n_1 and the end node of p is n_2). The node n_1 is a proper ancestor of n_2 (and n_2 is a proper descendant of n_1) if, in addition, $n_1 \neq n_2$. Given a node n of t, we use t^n_Δ to denote the subtree of t that is rooted at n.

We consider two types of rooted, labeled trees that represent XML documents and queries in XPath, respectively. An *XML tree* (*tree* for short) represents an XML document and its labels come from an infinite set Σ. We use $\mathbf{T}_\Sigma$ to denote the set of all the trees with labels from Σ. XPath queries are called *patterns* and they are different from XML trees in three aspects. First, the labels of a pattern come from the set $\Sigma \cup \{*\}$, where $*$ is the "wildcard" symbol ($* \in \Sigma$). Second, a pattern P has two types of edges: $\mathcal{E}_/(P)$ is the set of child edges and $\mathcal{E}_{//}(P)$ is the set of descendant edges. Third, a pattern P has an output node that is denoted by $out(P)$.

Patterns represent the fragment $XP^{\{//,[\,],*\}}$ of XPath that was extensively investigated in the past, and is described by the grammar

$$q \Rightarrow q/q \mid q//q \mid q[q] \mid l \mid *$$

where l is a label in Σ. Each pattern (constructed using the above grammar) has a tree-like form which is implied by parsing the pattern from left to right. The last label that we parse and it is not surrounded by square brackets describes the label of the output node of the pattern. For example, the pattern $P = a[b/d][*[e][f//g]]$ is illustrated in Figure 5.3.

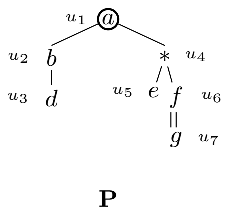


Figure 5.3: Tree-like form of the pattern $P = a[b/d][*[e][f//g]]$

The *selection path* of a nonempty pattern P in $XP^{\{//,[\,],*\}}$ is the path from the root to the output node. The nodes on the selection path are called *selection nodes*, while the edges on the selection path are called *selection edges*. The *depth* of a selection node v is the length of path from the root to v. For example, the depth of the root is 0. We usually denote the depth of the output node by d and we say that d is also the depth of the parrern P. For $0 \leq k \leq d$, the $k-$ *node* is the selection node at depth k. We extend the notion of depth as follows. For all nodes v of P, the depth of v is that of its deepest ancestor on the selection path. Moreover, considering a pattern P and a node n of P, the tree that comprises one child m of n (including the edge connecting n to m) and the subtree t^m_Δ is called a *branch* of n in t only if m is not a selection node. Observe that the number of branches of a node n is the number of children of n. Notice that each expression which is surrounded by square brackets in a pattern constructed using the above grammar represents a branch.

Consider a pattern P in $XP^{\{//,[\,],*\}}$ of depth d, and let k be an integer such that $0 \leq k \leq d$. The *k-sub-pattern* of P, denoted by $P^{\geq k}$, consists of all nodes v of P, such that the depth of v is greater than or equal to k. In other words, $P^{\geq k}$ is the subtree of P that is rooted at the k-node of P. The output node of $P^{\geq k}$ is that of P. The *k-upper-pattern* of P, denoted by $P^{\leq k}$, comprises all nodes of P at a depth of no more than k. That is,

$P^{\leq k}$ is obtained from P by pruning the subtree rooted at the $(k+1)$-node. The output node of $P^{\leq k}$ is the k-node of P. Note that $P^{\leq d}$ and $P^{\geq 0}$ are the same as P.

In addition, we refer to a *view* as a named pattern and we say that a view is *materialized* if its result is stored.

In this work we focus on the three major fragment of $XP^{\{//,[\,],*\}}$, which contain two of the constructs $*$, $//$ and branches ([ ]); i.e., on the fragments of $XP^{\{[\,],*\}}$, $XP^{\{//,[\,]\}}$ and $XP^{\{//,*\}}$. Notice, now, that each pattern in $XP^{\{//,*\}}$ describes a single selection path.

Figure 5.4 illustrates the pattern P in $XP^{\{//,*\}}$ whose depth is 4, and a tree t. Each node is represented by its unique name and its label; e.g. the node u_3 is labeled by c. The circled node indicates the output node of P (i.e. $out(P) = u_5$) and the top nodes of P and t indicate the root nodes of P and t, respectively (i.e. $root(P) = u_1$ and $root(t) = n_1$). We avoid to represent the direction of each edge of either tree or pattern; supposing that if a node m is a child of a node m' then m' appears above of m. Moreover, we represent the child edges with single lines and the descendant edges with double lines.

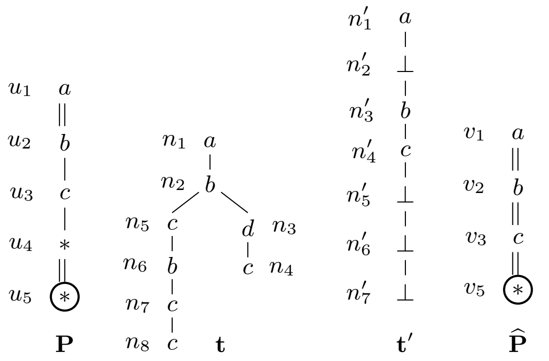


Figure 5.4: P is a pattern in $XP^{\{//,*\}}$

We, now, define a special class of patterns in $XP^{\{//,[\,],*\}}$, called *boolean patterns*, that are patterns which do not have any output node. In the following, we will see that the boolean patterns have a significant role on deciding containment and equivalence.

5.1.1 Semantics of Patterns

In this section, we define the result of applying a pattern on a tree and we describe a navigation on a tree through which we locate the result of a pattern.

Intuitively, a pattern specifies a set of conditions on the reachable paths of nodes of a tree. A node of a tree whose reachable path satisfies the conditions is included in the result of the pattern. In addition, the result of applying a pattern on a tree can be thought either as a set of references to nodes of the tree or as a set of subtrees that are extracted from the tree (since for each node u in a tree t the subtree t_{Δ}^{u} corresponds to an element of a document). Capturing both the above perspectives, we consider that the output of a pattern is a set of nodes that uniquely identifies a set of subtrees of a tree.

Typically, applying a pattern on a tree, we start by checking whether or not the root label of the pattern matches with the root label of the tree, and if so, we continue by searching the children of the root of the tree in order to match the labels of the children of the root of the pattern, and so on. The way that we are navigated inside the tree in order to match the labels of the nodes of the pattern is specified by the edges between the nodes of the pattern. For example, in Figure 5.4, applying the pattern P on the tree t, we firstly match the root labels (both P and t have the same root labels); then we search for a descendant of the root of t that is labeled by b (which is specified by the existence of the descendant edge from u_1 to u_2). We notice that the nodes n_2 and n_6 satisfy this condition. Then we continue by checking for each of these nodes whether there is any child labeled by c (as node u_3 is labeled by c). Notice that u_3 matches each of the children of the nodes n_2 and n_6; i.e., it matches the nodes n_5 and n_7, respectively. Then we focus on the children of n_5 and n_7. Since node u_4 is labeled by wildcard, it can match to any label. Continuing the same procedure, we conclude that the output node of P matches the nodes n_7 and n_8; which constitute the result of applying P on t.

As we noticed the result of a pattern is based on a set of mappings from the nodes of the patterns to the nodes of a tree such that the labels of the corresponding nodes match and each edge of the pattern maps on a path of a tree in a manner that a child edge of the pattern maps on an edge of the tree and a descendant edge of the pattern maps on a path of the tree. We formally define such a mapping from a pattern to a tree, called *embedding*, as follows.

Definition 14. An **embedding** from a pattern P in $XP^{\{//,[\,],*\}}$ to a tree

t is a mapping $e : \mathcal{N}(P) \to \mathcal{N}(t)$ with the following properties.

- Root preserving: $e(root(P)) = root(t)$.
- Label preserving: For all nodes $n \in \mathcal{N}(P)$, either $label(n) = *$ or $label(n) = label(e(n))$.
- Child preserving: For all edges $(n_1, n_2) \in \mathcal{E}_/(P)$, we have that $(e(n_1), e(n_2)) \in \mathcal{E}(t)$.
- Descendant preserving: For all edges $(n_1, n_2) \in \mathcal{E}_{//}(P)$, the node $e(n_2)$ is a proper descendant of the node $e(n_1)$.

Given an embedding $e : \mathcal{N}(P) \to \mathcal{N}(t)$, we denote by o_e the image of the output node, i.e. $o_e = e(out(P))$. The embedding e produces the tree $t_\Delta^{o_e}$, that is, the subtree of t that is rooted at o_e. We denote by $P(t)$ the result of applying the pattern P to the tree t; which is naturally defined as the set of subtrees produced by all embeddings from P to t. Formally, the result of applying a pattern P on a tree t is

$$P(t) = \{o_e | e\ is\ an\ embedding\ from\ P\ to\ t)\}$$

We also define the result of applying P on a set of trees T to be $P(T) = \bigcup_{t \in T} P(t)$. In addition, we refer to the reachable path of each node in $P(t)$ as a path which is *accepted* by P.

Here, we have to note that the result of applying a boolean pattern P on a tree t is either *true* or $false$. In order to compute the result $P(t)$ we also search for embeddings from P to t. If there is at least one embedding then $P(t)$ is *true*; otherwise $P(t)$ is $false$.

Practically, the result of applying a pattern on a tree is computed regarding an ordering of nodes of the tree. Hence, while the result of a pattern is a set, it may contain semantically duplicate trees; which, however, are considered distinct due to ordering of nodes of the tree. We refer to this type of semantics as *set semantics*; and throughout this work, we consider that each pattern is evaluated under set semantics (except if explicitly mentioned).

Considering, now, a set $\mathcal{P}$ of patterns in $XP^{\{//,[\,],*\}}$ and a tree t, we denote as $\mathcal{P}(t)$ the union of the results of applying all the patterns in $\mathcal{P}$ on t; i.e., $\mathcal{P} = \bigcup_{P \in \mathcal{P}} P(t)$. Here, we have to notice that we consider the conventional union operator which applies on sets and results a set (i.e., no duplicate trees w.r.t. the nodes-ordering, appear in the result of the union).

Moreover, considering that there is an embedding from a pattern P to a tree t, we say that t is a *model* of P. It is often useful to consider canonical models rather than general ones. Next, we define this type of models. We denote by $\perp$ a special label of Σ. Throughout this work, we assume that the patterns at hand do not include $\perp$ as a node label. A **canonical model** of a pattern P in $XP^{\{//,[\,],*\}}$ is a tree that is obtained from P by the following two steps.

1. Each occurrence of the label $*$ is replaced with $\perp$,
2. All the descendant edges are replaced by paths whose length is at least one and their internal nodes are labeled by $\perp$.

We use $Mod(P)$ and $CMod(P)$ to denote the set of all models and all canonical models of P, respectively. Moreover, we denote as $CMod_{\leq k}(P)$ the subset of $CMod(P)$ that contains all the canonical models obtained by replacing each descendant edge with a path whose length is at most k. Alternatively, we say that $CMod_{\leq k}(P)$ contains the canonical models of P that are *bounded by k*. Replacing, now, each descendant edge with a path of exact k edges, we get the *k-canonical model* of P, denoted as $CMod_k(P)$.

According to the construction of the k-canonical model of a pattern P, we use the names of the nodes of P to refer to the corresponding nodes in $CMod_k(P)$. In addition, we refer to the one-to-one embedding from P to $CMod_k(P)$ implied by the construction of $CMod_k(P)$ as the *k-canonical embedding* of P, denoted c_P^k, and to the image of the output node that is produced by c_P^k as *k-canonical output* of P, denoted o_P^k; i.e. $o_P^k = c_P^k(out(P))$. For example, in Figure 5.4 the tree t' is the $CMod_2(P)$ of the pattern P, where the circled node indicates the output of P. In addition, the c_P^2 maps the nodes u_1, u_2, u_3, u_4, u_5 on the nodes n'_1, n'_3, n'_4, n'_5, n'_7, respectively; and the o_P^2 is the subtree of t' rooted by the node n'_7.

5.1.2 Core Pattern

Focusing, now, on the fragment of $XP^{\{//,*\}}$, the conditions that a pattern specify on the reachable paths are of two types. Firstly, it requires both a set of labels to appear on each reachable path and the ordering of these labels in the paths. Secondly, it specifies limitations on the lengths of the paths between these labels. For example, in Figure 5.4, P requires that we have to start from a root labeled by a, then pass a node labeled by b and then pass a node labeled by c through the navigation on the reachable path of a resulted node (notice, here, that P does not provide any requirement on the label of the image of the output). In addition, P requires that the distance of the labels b and c in the reachable path of an output is exact 1 edge, while the distance of the label c and an output have to be at least 2 edges.

Therefore, it is easy to see that the root, the output and the non-wildcard labeled nodes of a pattern form a backbone of the paths accepted by the pattern. Consequently, we distinguish these nodes and refer to them as *core nodes* [CW10] of the pattern.

Definition 15. Let a pattern P in $XP^{\{//,*\}}$. We say that a node $u \in \mathcal{N}(P)$ is a **core node** if at least one of the following conditions hold:

1. v is the root of P, i.e. $v = root(P)$;
2. $label(v) \neq *$;
3. v is an output node, i.e. $v = out(P)$.

We use $\widehat{\mathcal{N}}(P)$ to denote the set of core nodes in $\mathcal{N}(P)$.

Remark 8. The previous definition implies that a node v in $\mathcal{N}(P)$ is not a core node of P only if it is labeled by a wildcard and it is an internal node. In addition, if there are two embeddings from P to a tree such that the images of a core node of P, using these embeddings, have different labels, then this core node is either the root or the output of the pattern and it is labeled by $*$.

In Figure 5.4, the $\widehat{\mathcal{N}}(P)$ contains the nodes u_1, u_2, u_3 and u_5.

Considering, now, the paths between two core nodes which contain only non-wildcard labeled nodes, it is easy to see that these paths describe limitations on the distances between the labels of the core nodes on each tree. For example, in Figure 5.4, for each embedding from P to a tree, the distance between the image $e(u_3)$ and the o_e is greater than or equal to 2 edges. We refer to such path as *core path* of a pattern [CW10] and we formally define it as follows.

Definition 16. Let a pattern P in $XP^{\{//,*\}}$. We say that a path $p = u, z_1, \ldots, z_n, v$ of P is a **core path**, where $n \geq 0$, if $z_1, \ldots, z_n \in (\mathcal{N}(P) - \widehat{\mathcal{N}}(P))$ and $u, v \in \widehat{\mathcal{N}}(P)$. The nodes u and v are called the *core endpoints* of p and the nodes $z_1, \ldots, z_n$ are called *intermediate nodes* of p.

The pattern P illustrated in Figure 5.4, has three core paths; which are: the path $p_1 = u_1, u_2$, the path $p_2 = u_2, u_3$ and the path $p_3 = u_3, u_4, u_5$.

Here, we have to notice that a core path can specify either the exact distance of the images of two core nodes (if all the edges in the core path are child edges) or the minimum acceptable distance (if there is at least one descendant edge in the core path).

In order to capture, now, the ordering of the images of core nodes in a tree, for each pattern we construct a unique pattern, called *core pattern*, which contains only the core nodes of the pattern, maintaining their ordering in the initial pattern. The core pattern is formally defined as follows.

Definition 17. Let a pattern P in $XP^{\{//,*\}}$. The **core pattern** of P, denoted $\widehat{P}$, is the pattern obtained from P such that $\mathcal{E}_/(\widehat{P}) = \emptyset$, $\mathcal{N}(\widehat{P}) = \widehat{\mathcal{N}}(P)$, and for each core path $u, z_1, \ldots, z_n, v$, where $n \geq 0$, of P we have that $(u, v) \in \mathcal{E}_{//}(\widehat{P})$.

In Figure 5.4, the edge v_1, v_2 of $\widehat{P}$ is obtained by the core path u_1, u_2 of P. Similarly, the core path u_2, u_3 gives the edge v_2, v_3; and the edge v_3, v_5 is obtained by collapsing the core path u_3, u_4, u_5.

5.2 Containment and Equivalence of sets of patterns

In this section, we define these problems of containment and equivalence of patterns and describe techniques which solve these problems for the fragment of $XP^{\{//,[\,],*\}}$ and the three major subfragments of it.

The containment and equivalence, intuitively, are two features which enable comparisons between patterns. Similarly to the relational model [AHV95, Ull88, Ull89, GMUW08], the containment (resp. equivalence) between patterns guarantees that the result of one pattern is always a part of (resp. the same to) the result of the other pattern; in the case that both patterns are applied on the same tree. Both features are formally defined as follows.

Definition 18. Let two patterns P_1, P_2 in $XP^{\{//,[\,],*\}}$. Then P_2 is **contained** in P_1, denoted $P_2 \sqsubseteq P_1$, if $P_2(t) \subseteq P_1(t)$ for all trees $t \in \mathbf{T}_\Sigma$. P_1 and P_2 are **equivalent**, denoted $P_2 \equiv P_1$, if $P_2 \sqsubseteq P_1$ and $P_1 \sqsubseteq P_2$, that is, $P_2(t) = P_1(t)$ for all trees $t \in \mathbf{T}_\Sigma$.

Therefore, the problems of containment and equivalence of the patterns P_1 and P_2 focus on deciding whether or not P_2 is contained or equivalent, respectively, to P_1.

Example 47. Consider the patterns P_1 and P_2 depicted on Figure 5.5. As the root nodes of P_1 and P_2 are also their ouput nodes, then, both patterns result the entire tree (on which they are applied). Notice that for both patterns, the resulted trees have to be rooted by an a-node. P_1 also requires that the root has a descendant labeled by g and a child labeled by c. Moreover, the node labeled by c must have a child labeled by e. On the other hand, P_2 requires that the root has a child labeled by c. This c-node is also required to have two children, one labeled by e and one which is labeled by f and it is a proper ancestor of a node labeled by g. Hence, each tree t which is contained in $P_2(t)$, also satisfies the requirements provided by P_1. Thus, this tree is also contained in $P_1(t)$. Consequently, for each tree t we have that $P_2(t) \subseteq P_1(t)$; which implies that $P_2 \sqsubseteq P_1$.

However, the opposite direction does not hold; i.e., $P_1 \not\sqsubseteq P_2$. To see this, consider the $CMod_1(P_1)$. Applying, now, both patterns on this tree, we have that $P_2(CMod_1(P_1)) = \emptyset$ and $P_1(CMod_1(P_1)) \neq \emptyset$. Hence, $P_1 \not\equiv P_2$.

□

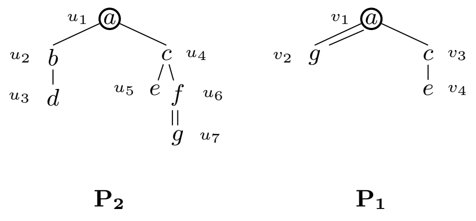


Figure 5.5: P_2 is contained in P_1

5.2.1 Deciding containment and equivalence of patterns

In this section we describe significant techniques used for deciding containment of two patterns. These techniques are categorized into two main classes. In the first class, the techniques focus on describing the minimum subset of models for each pattern such that the solutions of the problems are reduced to simple comparisons of these subsets. In the second one, the techniques are based on finding a mapping from one pattern to another through which each embedding of the contained pattern is translated to an embedding of the containing pattern.

Based on the definition of embedding (which actually describes a mapping between rooted, ordered trees), we define a mapping, called homomorphism, between nodes of patterns. The formal definition of pattern homomorphism is given as follows.

Definition 19. Let two patterns P_1, P_2 in $XP^{\{//,[\,],*\}}$. A **homomorphism** from P_1 to P_2 is a mapping $h : \mathcal{N}(P_1) \rightarrow \mathcal{N}(P_2)$ with the following properties.

- Root preserving: $h(root(P_1)) = root(P_2)$.
- Output preserving: $h(out(P_1)) = out(P_2)$.
- Label preserving: For all nodes $n \in \mathcal{N}(P_1)$, either $label(n) = *$ or $label(n) = label(h(n))$.
- Child preserving: For all edges $(n_1, n_2) \in \mathcal{E}_/(P_1)$, we have that $(h(n_1), h(n_2)) \in \mathcal{E}_/(P_2)$.
- Descendant preserving: For all edges $(n_1, n_2) \in \mathcal{E}_{//}(P_1)$, then node $h(n_2)$ is a proper descendant of the node $h(n_1)$.

We say that h is an **isomorphism** if h is a homomorphism from P_1 to P_2 and the mapping h^{-1} defined such that $h^{-1}(h(n)) = n$, for each node $n \in \mathcal{N}(P_1)$, is also a homomorphism from P_2 to P_1. In this case, we say that P_1 and P_2 are *isomorphic*, denoted as $P_1 \sim P_2$.

Notice that treating both homomorphism and embedding as mappings between rooted, ordered trees, a homomorphism is derived by enforcing the definition of the embedding with one more property (Output preserving). Hence, considering a homomorphism h from a pattern P_1 to a pattern P_2 (both P_1, P_2 are in $XP^{\{//,[\,],*\}}$), it is easy to verify that the composition of each embedding e_2 from P_2 to a tree t and h determines an embedding e_1 from P_1 to t such that $e_1(out(P_2)) = e_2(out(P_1))$. Thus, the existence of a homomorphism constitutes a sufficient condition for deciding containment of patterns in $XP^{\{//,[\,],*\}}$. The following proposition formally describes this result [MS04].

Proposition 24. *Let two patterns P_1, P_2 in $XP^{\{//,[\,],*\}}$. If there is a homomorphism from P_1 to P_2, then $P_2 \sqsubseteq P_1$.*

However, the existence of a homomorphism does not also constitute a necessary condition. The following example illustrates that even for the case that both patterns are in $XP^{\{//,*\}}$ the existence of homomorphism does not suffice for deciding containment.

Example 48. Let the patterns P_1 and P_2 in $XP^{\{//,*\}}$ depicted in Figure 5.6. Notice that both patterns search each tree rooted by an a-node for nodes labeled by b whose reachable paths contain at least 3 edges. Hence, $P_1 \equiv P_2$.

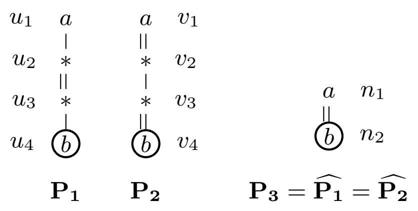


Figure 5.6: P_1 and P_2 are equivalent

However, it is easy to verify that there is not any homomorphism neither from P_1 to P_2 nor from P_2 to P_1. □

The question, now, that arises is whether there are fragments of $XP^{\{//,[\,],*\}}$ for which the existence of a homomorphism consist of a necessary and sufficient condition for deciding containment. The question is answered in [MS04, AYCLS01, Yan81, Woo01], where it is shown that the problem of containment for patterns either in $XP^{\{//,[\,]\}}$ or $XP^{\{[\,],*\}}$ is reduced to the problem of finding a homomorphism from candidate containing pattern to the candidate contained pattern. These result are formally summarized on the following proposition.

Proposition 25. *Let two patterns P_1, P_2 in $XP^{\{//,[\,]\}}$ and two patterns Q_1, Q_2 in $XP^{\{[\,],*\}}$. Then the following hold:*

- *$P_2 \sqsubseteq P_1$ **if and only if** there is a homomorphism from P_1 to P_2.*
- *$Q_2 \sqsubseteq Q_1$ **if and only if** there is a homomorphism from Q_1 to Q_2.*

Example 49. Considering the patterns P_1, P_2 introduced in Example 47, we notice that both P_1 and P_2 are in $XP^{\{//,[\,]\}}$. In addition, consider the mapping $h : \mathcal{N}(P_1) \to \mathcal{N}(P_2)$ such that v_1 maps on u_1, v_2 maps on u_7, v_3 maps on u_4 and v_4 maps on u_5. It is easy to verify that h is a homomorphism from P_1 to P_2; hence Proposition 25 implies that $P_2 \sqsubseteq P_1$. On the other hand, notice that there is no homomorphism from P_2 to P_1; which implies that $P_1 \not\sqsubseteq P_2$. □

Now, as we observed in Example 48, for the third fragment of $XP^{\{//,[\,],*\}}$ (i.e., for the fragment $XP^{\{//,*\}}$) a homomorphism does not suffice in order to decide containment. This is caused because the homomorphism is too strict for capturing the transfer of embeddings in the case that patterns have core paths with descendant edges and their lengths are greater than or equal to 2. More specifically, notice in Figure 5.6 that while there is no homomorphism from P_1 to P_2, every node n resulted by P_2 has reachable path which contains at least two internal nodes. This conditions suffice in order to exist at least one embedding from P_1 that maps $out(P_2)$ on n. Hence, for deciding containment in this fragment we define a new mapping between patterns, called d-homomorphism, by relaxing the concept of homomorphism. The formal definition is given as follows.

Definition 20. Let two patterns P_1 and P_2 in $XP^{\{//,*\}}$. We say that a homomorphism h from $\widehat{P_1}$ to $\widehat{P_2}$ is a **d-homomorphism** from P_1 to P_2 if for each edge (u, v) in $\mathcal{E}_{//}(\widehat{P_1})$, the following hold:

1. the length of the core path $u, \ldots, v$ in P_1 is less than or equal to the length of the path $h(u), \ldots, h(v)$ in P_2; and
2. if $u, \ldots, v$ in P_1 has only child edges then $h(u), \ldots, h(v)$ in P_2 has only child edges and its length is equal to the length of $u, \ldots, v$.

We say that h is a **d-isomorphism** if h is a $d-$homomorphism from P_1 to P_2 and the mapping h^{-1} defined such that $h^{-1}(h(n)) = n$, for each node $n \in \mathcal{N}(\widehat{P_1})$, is also a $d-$homomorphism from P_2 to P_1. In this case, we say that P_1 and P_2 are *d-isomorphic*, denoted as $P_1 \sim_d P_2$.

Corollary 5. *A $d-$homomorphism h from a pattern P_1 in $XP^{\{//,*\}}$ to a pattern P_2 in $XP^{\{//,*\}}$ is a $d-$isomorphism if h is an isomorphism from $\widehat{P_1}$ to $\widehat{P_2}$ and for each edge (u, v) in $\mathcal{E}_{//}(\widehat{P_1})$ such that $u, \ldots, v$ contains at least one descendant edge we have that $h(u), \ldots, h(v)$ has the same length with $u, \ldots, v$ and has at least one descendant edge.*

Remark 9. Here, we have to notice that each homomorphism between two patterns in $XP^{\{//,*\}}$ is also a d-homomorphism.

The following proposition formally shows that the existence of a d-homomorphism consists of a necessary and sufficient condition for deciding containment for patterns in $XP^{\{//,*\}}$ [MS99, MS04].

Proposition 26. *Let two patterns P_1 and P_2 in $XP^{\{//,*\}}$. $P_2 \sqsubseteq P_1$* ***if and only if*** *there is an d-homomorphism from P_1 to P_2.*

Here, we have to notice that in [MS99, MS04], the authors approach the above result as follows. Firstly, they replace the patterns with two new patterns. These patterns (called adorned tree patterns) are constructed from the initial ones by replacing all the core paths that have descendant edges with a single descendant edge associated with a number which corresponds to the length of the core path. Then the existence of a homomorphism-like mapping which also compares the adorned numbers of the descendant edges suffice in order to decide containment. It is easy to see the one-to-one correspondence of this mapping and the $d-$homomorphism.

Example 50. Continuing the Example 48, we construct the core patterns of P_1 and P_2. Notice that both patterns have the same core pattern P_3 which is also illustrated in Figure 5.6; i.e., $P_3 = \widehat{P_1} = \widehat{P_2}$. Considering, now, the isomorphism h from $\mathcal{N}(\widehat{P_1})$ to $\mathcal{N}(\widehat{P_2})$, we have that both conditions of the definition of d-homomorphism hold for h. Thus, h is a d-homomorphism. In addition, notice that the isomorphism h^{-1} from $\mathcal{N}(\widehat{P_2})$ to $\mathcal{N}(\widehat{P_1})$ is also d-homomorphism; consequently, h is a d-isomorphism. The Proposition 26, now, implies that both $P_2 \sqsubseteq P_1$ and $P_1 \sqsubseteq P_2$ hold; hence, $P_2 \equiv P_1$. □

Focusing, now, on comparing the models of the patterns, the definitions of the model and containment imply that deciding containment of patterns is equivalent to comparing the results of applying the patterns to the models of the candidate contained pattern. This result is formally described as follows.

Corollary 6. *Considering the patterns P_1, P_2 in $XP^{\{//,[\,],*\}}$, we have that $P_2 \sqsubseteq P_1$* ***if and only if*** *$P_2(Mod(P_2)) \subseteq P_1(Mod(P_2))$.*

However, as we can easily notice the set of models of a pattern in $XP^{\{//,[\,],*\}}$ is infinite; which concludes that we cannot focus on this condition for deciding containment.

Patterns in $XP^{\{//,[\,],*\}}$: In the case that both patterns are included in $XP^{\{//,[\,],*\}}$ neither a simple homomorphism nor a d-homomorphism suffices in order to decide containment [MS04]. The following example describes such a case.

Example 51. Let the patterns P_1 and P_2 in $XP^{\{//,[\,],*\}}$ illustrated in Figure 5.7. Notice that there is no homomorphism from P_1 to P_2. Moreover, constructing the core patterns $\widehat{P_1}$ and $\widehat{P_2}$ of P_1 and P_2 (see also in Figure 5.7), respectively, there are three homomorphisms from $\widehat{P_1}$ to $\widehat{P_2}$. Here, we have to notice that we retain the names of the nodes of the patterns to their core patterns. The first homomorphism from $\widehat{P_1}$ to $\widehat{P_2}$, denoted as h_1, maps for each $i = 1, 2, 3, 5, 6, 7, 8$, the node v_i on the node u_i. The second homomorphism from $\widehat{P_1}$ to $\widehat{P_2}$, denoted as h_2, maps v_1 on u_1, v_2 on u_2, v_3 on u_3, v_5 on u_5, v_6 on u_9, v_7 on u_{10} and v_8 on u_{11}. The third homomorphism from $\widehat{P_1}$ to $\widehat{P_2}$, denoted as h_3, maps v_1 on u_1, v_2 on u_6, v_3 on u_7, v_5 on u_8, v_6 on u_9, v_7 on u_{10} and v_8 on u_{11}. Moreover, notice that none of them is a d-homomorphism from P_1 to P_2; because h_1 and h_2 violate the second condition and h_3 violates the first condition of d-homomorphism. However, it is easy to verify that P_2 is contained in P_1. □

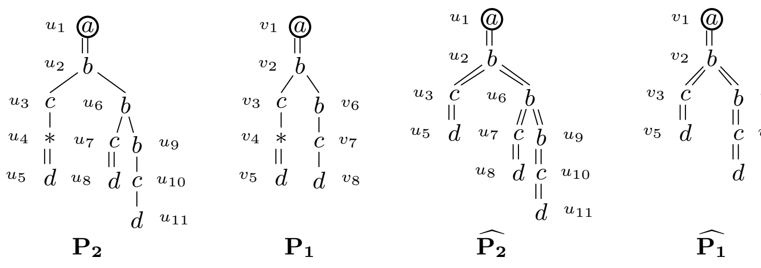


Figure 5.7: P_2 is contained in P_1

In this case, we focus on comparing the results of the patterns when they are applied on a set of models. More specifically, we search for a finite subset of models of the candidate contained pattern such that we can decide containment by applying both patterns on this subset and then comparing their results. Considering, now, the set of canonicals models of a pattern P in $XP^{\{//,[\,],*\}}$, notice that if a pattern has at least one descendant edge then $CMod(P)$ is infinite. Hence, we cannot use this subset to decide containment. However, we can focus on finding a bounded subset of the canonical models. In [MS04], it is shown that focusing on the paths that has only wildcard-labeled internal nodes, we can find such a subset. More specifically, considering the patterns P_1, P_2 in $XP^{\{//,[\,],*\}}$, we focus on the longest such path that appears either in P_1 or in P_2; and we denote its length as k_{max}. Constructing, now, the canonical models of P_2 that are bounded by k_{max}, we can decide the containment $P_2 \sqsubseteq P_1$ by comparing the results of applying both patterns on this set of models [MS04]. This result is formally described as follows.

Proposition 27. *Considering the patterns P_1, P_2 in $XP^{\{//,[\,],*\}}$, then the following are equivalent:*

1. $P_2 \sqsubseteq P_1$.
2. $P_2(CMod(P_2)) \subseteq P_1(CMod(P_2))$.
3. $P_2(CMod_{\leq k}(P_2)) \subseteq P_1(CMod_{\leq k}(P_2))$, *where $k = k_{max} + 1$, and k_{max} is the length of the longest core path appearing in P_1 and P_2.*

Example 52. Continuing the Example 51, notice that the longest paths are the paths u_3, u_5 and v_3, v_5; hence $k_{max} = 2$. Now, it is easy to verify that constructing the $CMod_{\leq 3}(P_2)$ (which contains 27 canonical patterns) and applying both P_1, P_2 on this we have that $P_2(CMod_{\leq 3}(P_2)) \subseteq P_1(CMod_{\leq 3}(P_2))$; hence, $P_2 \sqsubseteq P_1$. On the other hand, constructing the $CMod_{\leq 3}(P_1)$ (which contains 9 canonical patterns) and applying both P_1, P_2 on this we have that $P_1(CMod_{\leq 3}(P_1)) \nsubseteq P_2(CMod_{\leq 3}(P_1))$; hence, $P_1 \not\sqsubseteq P_2$. □

In [MS04], it is also proved that the containment problem of two patterns in $XP^{\{//,[\,],*\}}$ can be translated into deciding the containment of two boolean patterns. We say that a boolean pattern P_2 is contained in the boolean pattern P_1, also denoted $P_2 \sqsubseteq P_1$, if for every t such that $P_2(t) = true$ then $P_1(t)$ is also $true$.

Consider the patterns P_1, P_2 in $XP^{\{//,[\,],*\}}$ and a label s which is not included in Σ. We, now, construct two boolean patterns, denoted by P_1^B and P_2^B respectively, from P_1 and P_2 by adding in their output nodes a child node labeled by s . This procedure is called *boolean translation* of the patterns P_1 and P_2. The following proposition shows that deciding containment of P_1^B and P_2^B suffice in order to decide containment between P_1 and P_2.

Proposition 28. *Let two patterns P_1, P_2 in $XP^{\{//,[\,],*\}}$ and P_1^B, P_2^B be the patterns obtained using the boolean translation over P_1 and P_2, respectively. $P_2 \sqsubseteq P_1$* ***if and only if*** *$P_2^B \sqsubseteq P_1^B$.*

Equivalence of patterns: The definition of equivalence implies that the decision of the equivalence between two patterns is given by deciding containment in both ways. Hence, considering two patterns P_1 and P_2 we can check whether or not P_1 is equivalent to P_2 by using the techniques referred previously in order to decide whether both $P_2 \sqsubseteq P_1$ and $P_1 \sqsubseteq P_2$ hold.

In addition, the [ACG$^+$09] gives sufficient conditions for the equivalence of patterns in $XP^{\{//,[\,],*\}}$. These conditions describe some basic properties of the selection paths of equivalent patterns. More specifically, considering two equivalent patterns P_1 and P_2 in $XP^{\{//,[\,],*\}}$, the definition of the equivalence implies that every node of a tree which is an image of the $out(P_1)$ through an embedding e_1, is also an image of the $out(P_2)$ through an embedding e_2. Hence, the length of the reachable path of each such node is less than or equal to the depths of both P_1 and P_2. It is easy to see that this implies that the depths of P_1 and P_2 are equal. Moreover, using the same approach we can easily prove that the nodes appearing in the same depth in the selection paths of P_1 and P_2 have the same label. Both results are formally described by the following proposition (which summarizes the cases 1 and 3 of the Proposition 3.1 in [ACG+09]).

Proposition 29. *Let P_1 and P_2 be two equivalent patterns in $XP^{\{//,[\,],*\}}$ with depths d_1 and d_2, respectively. For all k, where $0 \leq k \leq d_1$, the following hold.*

1. *$d_1 = d_2$; and*
2. *the k-nodes of P_1 and P_2 have the same label.*

5.2.2 Containment of unions

The features of containment and equivalence of patterns defined in the previous section are straightforwardly extended to sets of patterns. More specifically, for two given sets $\mathcal{Q}_1$, $\mathcal{Q}_2$ of patterns in $XP^{\{//,[\,],*\}}$ we want to decide whether or not the patterns in $\mathcal{Q}_2$ result either a part of or the same information resulted by the patterns in $\mathcal{Q}_1$. Deciding the existence of these features between two sets of patterns we use the union operator over the results of the patterns of each set. Therefore, we formally define the containment and equivalence between unions of patterns as follows.

Definition 21. Let two sets $\mathcal{Q}_1$, $\mathcal{Q}_2$ of patterns in $XP^{\{//,[\,],*\}}$. Then $\mathcal{Q}_2$ is **contained** in $\mathcal{Q}_1$, denoted $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$, if $\mathcal{Q}_2(t) \subseteq \mathcal{Q}_1(t)$ for all trees $t \in \mathbf{T}_\Sigma$. $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are **equivalent**, denoted $\mathcal{Q}_2 \equiv \mathcal{Q}_1$, if $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ and $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$, that is, $\mathcal{Q}_2(t) = \mathcal{Q}_1(t)$ for all trees $t \in \mathbf{T}_\Sigma$.

We can equivalently say that the union $\bigcup_{P_2 \in \mathcal{Q}_2} P_2$ is either equivalent to or contained in the union $\bigcup_{P_1 \in \mathcal{Q}_1} P_1$. The following example shows a case where a set of patterns is contained in a single pattern.

Example 53. Consider the patterns P_1, P_2 and P_3 illustrated in Figure 5.8. Here, we notice that there is a homomorphism from P_1 to P_2; which implies that $P_2 \sqsubseteq P_1$. However, neither P_1 is contained in P_3 nor P_3 is contained in P_1. This can be verified by constructing the 1-canonical models of P_1 and P_3. Applying P_3 on $CMod^1(P_1)$ we can easily see that $P_3(CMod^1(P_1)) = \emptyset$; the same result holds when we apply P_1 on $CMod^1(P_3)$, i.e., $P_1(CMod^1(P_3)) = \emptyset$.

However, studying further the relationship between P_1 and P_3 we have that each reachable path of length greater than 3 which is accepted by P_1 is also accepted by P_3. In order to verify this, notice that P_3 accepts every path with start node labeled by a and end node labeled by b, which is also has length at least 3 edges. On the other hand, P_1 requires the reachable path of each node in its result to have length at least 2 edges and the same start and end nodes. It is to see, now, that the reachable paths that are accepted by P_1 and not accepted by P_3, are also accepted by P_1. More specifically, for an arbitrary path p which is accepted by P_1, if the length of p is equal to 2 edges then it is accepted by P_2; otherwise it is accepted by P_3. Consequently, we can see that for every tree t, we have that $P_1(t) \subseteq P_2(t) \cup P_3(t)$. Hence, $P_1 \sqsubseteq P_2 \cup P_3$. □

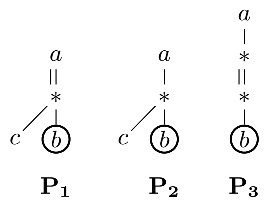


Figure 5.8: $P_1 \sqsubseteq P_2 \cup P_3$

The above example shows that the containment of unions of patterns in $XP^{\{//,[\,],*\}}$ can not straightforwardly decided by checking the containment of the patterns included in the unions. In [MS04], however, the authors prove that properly translating the patterns in both unions into patterns in $XP^{\{//,[\,],*\}}$ we can decide the containment of unions. Moreover, the above translation requires that the patterns of both unions are translated into boolean patterns (using the boolean translation).

Consider the sets $\mathcal{Q}_1$, $\mathcal{Q}_2$ of patterns in $XP^{\{//,[\,],*\}}$. Moreover, let an arbitrary pattern $P \in \mathcal{Q}_2$ and $\mathcal{Q}_1 = \{Q_1, \ldots, Q_k\}$. Without loss of generality we consider that all patterns of $\mathcal{Q}_1$ have the same root label (if not, we transform the patterns by adding another root node such that all the new patterns have the same root label). We construct, now, the patterns P_0 and Q_0 from P and $\mathcal{Q}_1$, respectively, as follows. Let two paths p_1 and p_2 whose start nodes are labeled by $r \in \Sigma$ and all other nodes are labeled by $c \in \Sigma$ (r and c are arbitrarily chosen). Moreover, p_2 has only child edges and its length is $2(k-1)+1$. p_1 has length equal to k and its only descendant edge connects its start node with its child. The pattern Q_0 is obtained by adding the patterns $Q_1, \ldots, Q_k$ as subtrees to the nodes of p_1. More specifically, for every $i = 1, \ldots, k$, we connect using a child edge the root of Q_i with the c-node appearing in depth i. Q_0 is illustrated in Figure 5.9. On the other hand, P_0 is obtained from p_2 by adding P as subtree of the node appearing in depth k. Moreover, we find a pattern V such that for any i , $V \sqsubseteq Q_i$, and we add it as subtree to any other non-root node of p_2. P_0 is also illustrated in Figure 5.9. The pattern V can be achieved by fusing the (common) roots of the Q_i subtrees (this is possible because their roots have the same label), and replacing all wildcards in the Q_i with an arbitrary letter, and all descendant edges with child edges.

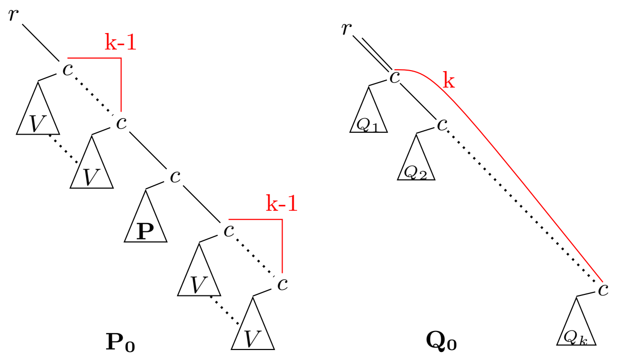


Figure 5.9: Translation of patterns

Using, now, the above translation of P and $\mathcal{Q}_1$, we can decide whether or not the containment $P \sqsubseteq \mathcal{Q}_1$ holds by deciding whether the pattern P_0 is contained in Q_0. More specifically, for every embedding e from P to an arbitrary tree t we can obtain a boolean pattern P_0' by replacing the P-subtree in P_0 with the subtree $e(P)$. Since from every $Q_i \in \mathcal{Q}_1$ there is a homomorphism to V then we can easily see that there is a homomorphism from Q_0 to P_0' if and only if there is an embedding from some pattern in $\mathcal{Q}_1$ to $e(P)$. Consequently, we can use the above translation to construct for every two sets $\mathcal{Q}_1$, $\mathcal{Q}_2$ of patterns in $XP^{\{//,[\,],*\}}$ two new sets $\mathcal{Q}_1'$, $\mathcal{Q}_2'$ of patterns also in $XP^{\{//,[\,],*\}}$ such that the decision of the containment $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ is reduced on deciding containment of each pattern of $\mathcal{Q}_1'$ and a pattern of $\mathcal{Q}_2'$. The following proposition describes this result [MS04, ZLWL09].

Proposition 30. *Let the sets $\mathcal{Q}_1$, $\mathcal{Q}_2$ of patterns in $XP^{\{//,[\,],*\}}$. Then there exist two sets $\mathcal{Q}_1'$, $\mathcal{Q}_2'$ of patterns in $XP^{\{//,[\,],*\}}$ such that $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ **if and only if** for each pattern $P_2 \in \mathcal{Q}_2'$ there is a pattern P_1 in $\mathcal{Q}_1'$ such that $P_2 \sqsubseteq P_1$.*

In the case, now, that patterns in both sets $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are in $XP^{\{//,[\,]\}}$, the decision of the containment is much more easier. More specifically, In [ZLWL09], it is proved that checking the containment between every pattern of $\mathcal{Q}_2$ and a pattern in $\mathcal{Q}_1$ suffices in order to decide whether the containment

$\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ holds. This result is formally given by the following proposition [ZLWL09].

Proposition 31. *Let the sets $\mathcal{Q}_1$, $\mathcal{Q}_2$ of patterns in $XP^{\{//,[\,]\}}$. Then $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$* ***if and only if*** *for each pattern $P_2 \in \mathcal{Q}_2$ there is a pattern P_1 in $\mathcal{Q}_1$ such that $P_2 \sqsubseteq P_1$.*

The above result can be easily verified by considering an embedding from a pattern in $\mathcal{Q}_1$ to the 2-canonical model of an arbitrary pattern in $\mathcal{Q}_1$. By the definition, however, of embedding we conclude that in the case that both patterns are in $XP^{\{//,[\,]\}}$ this embedding determines a homomorphism from one pattern to another. Hence since deciding the containment of two patterns in $XP^{\{//,[\,]\}}$ is achieved by finding a homomorphism, this suffices in order to show the above result.

For the other two major fragments of $XP^{\{//,[\,],*\}}$ (i.e., $XP^{\{//,*\}}$ and $XP^{\{[\,],*\}}$), the containment of unions is studied in next chapter.

5.2.3 Complexity of containment problem

In this paragraph we describe the complexities of both computing the result of applying a pattern on a tree and deciding containment for each fragment of $XP^{\{//,[\,],*\}}$ described previously.

Considering the size of a pattern P in $XP^{\{//,[\,],*\}}$, denoted $|P|$, as the number of edges appearing in P, we can find all the embeddings from P to a tree t in $O(|P||t|)$ time [MS04]. The algorithm introduced in [MS04] is based on, firstly, mapping the leaf nodes of P on nodes of t and then, using a leaf-to-root pruning of the tree. Using a similar algorithm the authors also prove that we can decide containment for each fragment of $XP^{\{//,[\,],*\}}$ that contains at most two of the constructs $*$, $//$ and branches ($[\,]$), in $PTIME$. More specifically, considering two patterns P_1 and P_2 in any of the previous fragments, we can decide whether there is either any homomorphism or any d-homomorphism in $O(|P_1||P_2|)$ time. However, considering the patterns in $XP^{\{//,[\,],*\}}$, the complexity of the containment problem jumps to $coNP$. More specifically, deciding the containment between two patterns in $XP^{\{//,[\,],*\}}$ is $coNP$-complete [MS04]; which is implied by the construction of the required canonical models as Proposition 27 shows. These complexity results, together with the results introduced in Section 6.1.3, are summarized in Table 5.1.

Moreover, in [MS04], the authors also prove that the problem is $coNP$-complete even in the case that the candidate contained pattern has at most

Patterns	Complexity	References
$XP^{\{[\,],*\}}$	$PTIME$	[MS04, Woo01, Yan81]
$XP^{\{//,[\,]\}}$	$PTIME$	[MS04, AYCLS01]
$XP^{\{//,*\}}$	$PTIME$	[MS04, MS99]
$XP^{\{//,[\,],*\}}$	$coNP$-complete	[MS04]
Union of $XP^{\{//,[\,],*\}}$	$coNP$-complete	[MS04]
Union of $XP^{\{//,*\}}$	in $coNP$	[MS04], Theorem 20
Union of $XP^{\{[\,],*\}}$	$PTIME$	Theorem 19
Union of $XP^{\{//,[\,]\}}$	$PTIME$	[ZLWL09]

Table 5.1: Complexity of containment of XPath fragments

5 path-branches (i.e., each branch is a single path) and the candidate containing patterns has at most 3 path-branches. The same complexity result holds in the case of bounded number of wildcard labeled nodes (they prove the case that both patterns contain at most 3 wildcard labeled nodes).

Deciding, now, the containment of unions of patterns in $XP^{\{//,[\,],*\}}$, the complexity is similar. Due to the result described in Proposition 30, we can decide this containment by properly translating both unions. The size of new patterns in both unions is polynomial to the size of the patterns included in the initial unions. Hence, the complexity of this problem is also $coNP$-complete. We can decide, however, the containment of unions when all patterns in both unions are in $XP^{\{//,[\,]\}}$, in $PTIME$ [ZLWL09]. This result is implied by Proposition 31.

5.2.4 Further related work on containment and equivalence of patterns

Containment and equivalence for patterns in $XP^{\{//,[\,],*\}}$ was further investigated in [CW10], where bag semantics are introduced for the evaluation of patterns. Computing the result of applying a pattern on a tree under bag semantics, multiple occurrences of a node (i.e., of a specific subtree) are allowed in the result. For example, applying the pattern $P = */*[//zoom]$ under set semantics on the tree illustrated in Figure 5.2, the element *Sony* appears once in the result of P. However, considering bag semantics the element *Sony* appears twice in the result of P, since there are two descendant elements of *Sony* labeled by *zoom*. In this work, it is also proved that the problem of deciding containment of two patterns in $XP^{\{//,[\,],*\}}$ under bag semantics is undecidable. The same result holds for the containment of multisets of patterns. However, they find necessary and sufficient conditions for the decision of equivalence of multisets of patterns. In this result the authors consider that each tree is associated with a partial ordering of sibling nodes. If, now, the patterns are not completely ordered (i.e., it is not necessary the partial ordering of sibling nodes), then deciding equivalence of two multisets is in $PSPACE$. These results are based on a transformation of each pattern, similar to that introduced in next chapter.

In [ZLWL09, CDO08], it is studied the problem of deciding equivalence between two intersections of $XP^{\{//,[\,]\}}$ patterns. In this case, the authors shows that an intersection of patterns can be translated to an equivalent union of patterns in $XP^{\{//,[\,]\}}$. Moreover, they show that the equivalence between a pattern and a union of patterns in $XP^{\{//,[\,]\}}$ holds if and only if the pattern is equivalent to a pattern of the union and all other patterns are contained to it. Hence, they showed that for patterns in this fragment the decision of equivalence between two intersections is properly reduced to deciding whether two unions of patterns are equivalent.

Deciding, now, containment in the presence of constraints is also investigated [Woo03, DT01, NS03]. As one might expect, when we consider containment of queries under constraints, the complexity increases. Given a DTD D, deciding containment under D (D-containment for short), even for queries in $XP^{\{[\,]\}}$, is coNP-complete [NS03, Woo01]. Containment is undecidable when the XPath fragment includes $XP^{\{//,[\,],*\}}$ along with disjunction, variable binding and equality testing, and the constraints include so-called bounded simple XPath integrity constraints (SXICs) and those (unbounded) constraints implied by DTDs [DT01]. In [Woo03], however, it is proved that D-containment is decidable (EXPTIME-complete, in fact) for $XP^{\{//,[\,],*\}}$. The decision of this containment is achieved using a translation of both patterns and DTD to tree grammars. Hence, comparing the corresponding grammars we decide whether or not the containment holds.

5.2.5 Answering a pattern using a single view

In this section we focus on describing the basic concepts of rewriting a pattern in $XP^{\{//,[\,],*\}}$ using views. Rewriting patterns is a fundamental technique of answering patterns using views. Similarly to the relational model [Hal01], the problem of answering a pattern using views can be stated as follows: given a set of patterns in $XP^{\{//,[\,],*\}}$ and a set of views also in $XP^{\{//,[\,],*\}}$, we want to find the result of the pattern using only the trees resulted by the views.

The technique of rewriting a pattern using views is based on finding a new pattern such that applying it on the trees resulted by the view its results are related to the results of the initial pattern. Formally, the *rewriting problem* can be stated as follows: given a pattern P and a views V both in $XP^{\{//,[\,],*\}}$, we want to find a new pattern R in $XP^{\{//,[\,],*\}}$ such that for every tree t, $P(t)$ is either equal to $R(V(t))$ or $R(V(t))$ is a subset of $P(t)$. We say that the pattern R is an *equivalent rewriting* (resp. a *contained rewriting*) of P if $R(V(t)) = P(t)$ (resp. if $R(V(t)) \subseteq P(t)$). In addition, we use the shorthand of *rewriting* to refer to an equivalent rewriting. The *rewriting-existence* problem, now, is that of determining, for a pattern P and a view V, whether there is a rewriting R of P using V.

Example 54. Let the patterns P, R_1 and the view V_1 illustrated in Figure 5.10. The view V_1 results the nodes which have a grandchild labeled by d and they are descendants of a root node labeled by a. On the other hand, R_1 results nodes labeled by c which are grandchildren of an arbitrary labeled root, have parent labeled by b and a sibling labeled by d. Notice, now, that applying V_1 on an arbitrary tree t and then applying R_1 on the result of V_1 we always get a result equal to $P(t)$. This shows that R_1 is a rewriting of P using V_1. □

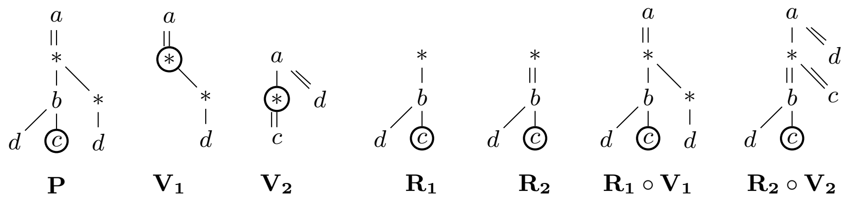


Figure 5.10: Rewritings of a pattern

The definitions of contained and equivalent rewriting imply a significant property for relationship of the rewriting pattern and the view. This property is formally defined in the following paragraph, and is based on the concept of pattern composition.

Pattern Composition: The greatest lower bound of two labels l_1 and l_2, denoted by $glb(l_1, l_2)$, is defined as follows. If $l \in \Sigma \cup \{*\}$ then $glb(l, l) = glb(l, *) = glb(*, l) = l$. If $l_1, l_2 \in \Sigma$ and $l_1 \neq l_2$, then $glb(l_1, l_2) = \diamond$ (where $\diamond \neq \Sigma$).

The composition of a pattern R with a pattern V, denoted by $R \circ V$, is obtained as follows. Let l_R^r be the label of the root of R and let l_V^o be the label of the output node of V. If $glb(l^r_R, l^o_V) = \bot$, then $R \circ V = \Upsilon$ (the empty pattern). Otherwise, $R \circ V$ is obtained by merging the output node of V with the root of R and as signing the label $glb(l^r_R, l^o_V)$ to the merged node. Note that the children of the merged node are all those of $out(V)$ and $root(R)$. The pattern $R \circ V$ has the same root as V and the same output node as R. As a special case, if $root(R) = out(R)$, then the merged node is the output node of $R \circ V$.

In previous work, it is shown that applying $R \circ V$ to a tree is the same as first applying V and then applying R to t.

Proposition 32. *$R \circ V(t) = R(V(t))$ holds for all trees $t \in \boldsymbol{T}_\Sigma$.*

Example 55. Continuing the Example 54 the composition of the pattern R_1 and the view V_1 is also illustrated in Figure 5.10. It is easy to see that the patterns $R_1 \circ V_1$ and P are identical. Hence, their equivalence and Proposition 32 also imply that R_1 is a rewriting of P using V_1. □

Natural rewriting candidates: In Example 54, V_1 is identical to $P^{\leq 1}$. Hence, V_1 and $P^{\leq 1}$ are equivalent. Moreover, notice that R_1 is identical to $P^{\geq 1}$. It is easy to see, now, that if a view V is equivalent to $P^{\leq k}$, for some non-negative integer k and a pattern P, then there is at least one rewriting of P using V; and one of them is given by the pattern $P^{\geq k}$. Can we, however, solve in any case the rewriting-existence problem by checking all the sub-patterns of a pattern? The answer is negative and is given by the following example.

Example 56. Let the pattern P and the view V_2 illustrated in Figure 5.10. Considering, now, the pattern R_2 which is also illustrated in Figure 5.10, we can see that R_2 is not a sub-pattern of P. However, the $R_2 \circ V_2$ is equivalent to P; which means that R_2 is a rewriting of P using V_2. □

This example leads us to the definition of *natural candidates*.

Let Q be a pattern in $XP^{\{//,[\,],*\}}$. We use $Q_{r//}$ to denote the pattern that is obtained by relaxing the edges that emanate from the root of Q. Observe that $Q \sqsubseteq Q_{r//}$. Now, consider a pattern P and a view V with depths d_P and d_V, respectively. The pattern R is a *natural rewriting candidate* (or just *natural candidate*) w.r.t. P and V if R is either $P^{\geq k}$ or$P^{\geq k}_{r//}$. The pattern R is a *potential rewriting* w.r.t. P and V when the following condition holds: If there is some rewriting, then R is also a rewriting; in other words, if R is not a rewriting, then one does not exist.

In [XO05], it is proved that for each of the three major fragments of $XP^{\{//,[\,],*\}}$ (i.e., the pattern, the view and the rewriting are together either in $XP^{\{//,*\}}$ or in $XP^{\{[\,],*\}}$, or in $XP^{\{//,[\,]\}}$) the natural candidates of the pattern suffice in order to solve the rewriting-existence problem. The following theorem formally describes this result.

Theorem 18. *Let a pattern P and a view V. If both P and V are either in $XP^{\{//,*\}}$ or in $XP^{\{[\,],*\}}$, or in $XP^{\{//,[\,]\}}$ then at least one of the natural candidates is a potential rewriting.*

In each of the three above cases, the rewriting-existence problem is solved in $PTIME$, since we can find all the natural candidates of a pattern in $PTIME$ and check the required equivalence also in $PTIME$.

For the case, now, in which P and V are both in $XP^{\{//,[\,],*\}}$, the authors in [XO05], shows that the rewriting-existence problem is $coNP$-hard. However, they do not give a tight complexity bound. In [ACG+09], it is studied several cases in which at least one of the natural candidates is a potential rewriting. For most of them the rewriting-existence problem is $coNP$-complete; which is implied by the complexity of deciding equivalence of two $XP^{\{//,[\,],*\}}$ patterns.

Further related work on rewriting patterns: The rewriting-existence problem is also studied in [BÖB+04, CR02, MS05, YLH03]; where incomplete algorithms which focus on answering patterns using cashed views are proposed. The problem of finding maximally contained rewritings, either in the absence or presence of a schema, is studied in [LWZ06] for the fragment $XP^{\{//,[\,]\}}$ (i.e., without wildcards). Moreover, the query answering using views has been studied extensively for the class of regular path queries [CGLV00, GT03] and in semistructured databases [PV99]. In [DT03], the problem of query reformulation for XML publishing is stated and solved in a general setting that allows both XML and relational storage for the data. In [KS08], the notions of redundancy and minimization are explored for the same fragment of XPath we study in this work. However, unlike the case of conjunctive queries, results on rewriting XPath queries are not easily derived from what is known about minimization of those queries.

The problem of rewriting a pattern using multiple views is also investigated in [CDO08, CDOV09, WY08, TYÖ+08, MS05]. In [CDO08, CDOV09], it is studied the problem of equivalently rewriting a pattern in $XP^{\{//,[\,]\}}$ using an intersection of views. The same problem is also studied in [WY08]. In addition, the authors in [WY08] use intersections of pattern to find maximal contained rewritings. On other hand, rewritings using multiple views are used for solving the view selection problem [TYÖ+08, MS05]. In [TYÖ+08], it is proposed a Nondeterministic Finite Automata (NFA) based approach to filter views that cannot be used to answer a query, and consequently obtain a candidate view set which equivalently answers a given pattern. In [MS05], a different approach is proposed for selecting views in order to answer a given pattern. This approach focus on finding common subexpressions of patterns.

Chapter 6

Answering patterns using union rewritings

As information on the Web increases rapidly, the need for substantial work on optimizing queries that manipulate XML data arises. In previous chapters, we were referred to techniques of rewriting queries in order to answer queries using views. This approach consists of one of the fundamental problems in databases with practical applications in data exchange, information integration, query optimization, data warehousing and Web-site design. Moreover, since XML data are popular for data exchange as well as for representing and manipulating semistructured data, rewriting XML queries become a significant problem.

Most of early research on rewriting problem for XPath patterns study the problem of finding a rewriting of a pattern using a single view. However, there are many cases in which single-view rewritings have limited benefit; since a single pattern can be equivalently rewritten using multiple single-view rewritings. The following example describes such a case.

Example 57. Consider the pattern P and the views V_1 and V_2, illustrated in Figure 6.1. P accepts the paths in which a b-node is a descendant of an a-labeled root. Moreover, each accepted path has resulted node which is labeled by c and is a grandchild of the b-node. V_1, now, results each b-node which is a child of the a-labeled root. Moreover, V_2 accepts the paths that are accepted by P and in which the root of the tree is at least 2 edges far from the b-node. We can easily notice, now, that there is no pattern R in $XP^{\{//,*\}}$ such that applying R on the result of one of the above views we equivalently answer P. However, for each view we can find at least one single-view contained rewriting. In order to verify this, consider the patterns

$R_1 = */*/c$ and $R_2 = c$. Applying the pattern R_1 on the results of V_1 and R_2 on the results of V_2, we get the two contained rewritings of P. However, we can notice that the result of R_1 captures the accepted paths of P in which the b-node is child of the root, and the R_2 accepts every other path which is accepted by P. Thus, using the patterns $R_1 \circ V_1$ and $R_2 \circ V_2$ we can equivalently answer P using only the answers of views; more specifically, the union $R_1 \circ V_1 \cup R_2 \circ V_2$ equivalently rewrites P. □

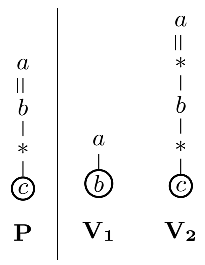


Figure 6.1: A single-view rewriting does not suffice

In this chapter, we focus on describing how a pattern can be equivalently rewritten using unions of single-view rewritings. For this purpose, we further investigate the problems of containment and equivalence for unions of patterns (Section 6.1). Considering patterns in fragment of $XP^{\{[\,],*\}}$ we give a necessary and sufficient condition for containment. Moreover, using this condition, we decide the containment of two unions in polynomial time (Section 6.1.1). Then, we investigate the problem of containment of unions for the case in which all patterns are in $XP^{\{//,*\}}$ (Section 6.1.3). For this fragment, we also give a necessary and sufficient condition for containment. The problem of equivalence of a pattern and either a single pattern or a union of patterns is studied in Sections 6.1.2 and 6.1.4, respectively. In the last section, we focus on finding an equivalent rewriting (called union rewriting) of a pattern using multiple views (Section 6.2); which is constructed using unions of single-view rewritings. This problem is investigated for the case in which the pattern, the rewritings and the views are in $XP^{\{//,*\}}$.

6.1 Containment and equivalence of unions of patterns

Contrary to the unions of CQs in the relational model [AHV95], deciding containment of sets of patterns is not reduced on deciding containment of single patterns. More specifically, we cannot decide containment of two sets by checking whether or not each pattern in the candidate contained set is contained to a pattern appearing in the candidate containing set. The following example illustrates such a case.

Example 58. Consider the patterns P_1, P_2, P_3 and P_4 in Figure 6.2. On checking whether or not the set $\mathcal{Q}_2 = \{P_3, P_4\}$ is contained in the set $\mathcal{Q}_1 = \{P_1, P_2\}$, we notice that there is a d-homomorphism from P_2 to P_3 but there is not any from either P_1 or P_2 to P_4. Hence, Proposition 26 implies that $P_3 \sqsubseteq P_1$ but $P_4 \not\sqsubseteq P_1$ and $P_4 \not\sqsubseteq P_2$. However, considering an arbitrary node n in the result of P_4 we notice that if the reachable path of n is of length 4 then n is resulted by P_1. Otherwise, if the length of the reachable path is greater than 4 then n is resulted by P_2. Hence, for every tree t we have that $P_3(t) \cup P_4(t) \subseteq P_1(t) \cup P_2(t)$; which implies containment. □

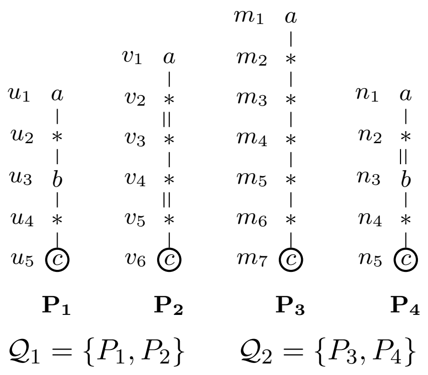


Figure 6.2: $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$

The question that arises is whether there is any fragment of $XP^{\{//,[\,],*\}}$ such that the containment of two sets of patterns can be decided by simply checking the containment between their patterns. As we referred in Section 5.2.2, this question is answered positively in [ZLWL09], for the fragment of $XP^{\{//,[\,]\}}$. What happens, however, for the cases in which the patterns are either in $XP^{\{[\,],*\}}$ or $XP^{\{//,*\}}$. As we showed in the Example 58 the above property does not hold for the second fragment. The solution of the problem, in this case, is studied in Section 6.1.3 and depends on the equivalence between two single patterns; which is studied in Section 6.1.2. The former fragment is studied in the following section.

6.1.1 Deciding unions containment in $XP^{\{[\,],*\}}$

Considering patterns in $XP^{\{[\,],*\}}$, each such pattern does not have any descendant edge. Therefore, it has a single canonical model. Considering, now, an embedding from a pattern P_1 to the canonical model of a pattern P_2, we can easily notice that if this embedding maps the output of P_1 on the canonical output of P_2 then it determines a homomorphism from P_1 to P_2. This observation consists of the key for proving that deciding containment of sets of such patterns is reduced on checking the containment of their patterns. The following theorem proves this statement.

Theorem 19. *Let two sets $\mathcal{Q}_1$, $\mathcal{Q}_2$ of patterns in $XP^{\{[\,],*\}}$. $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$* ***if and only if*** *for each pattern $P_2 \in \mathcal{Q}_2$ there is pattern $P_1 \in \mathcal{Q}_1$ such that $P_2 \sqsubseteq P_1$. We decide whether or not the $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ holds in $PTIME$.*

Proof. **(If part)** The proof of this part immediately follows from the definition of the union of sets.

(Only-If part) Consider that $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$. Moreover, let P_2 be a pattern in $\mathcal{Q}_2$. Since $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$, we have that if o is the image of $out(P_2)$ on a tree t using an embedding from P_2 then $o \in \mathcal{Q}_1(t)$. Considering, now, the canonical model t_c of P_2, its canonical output o_c is also contained in $\mathcal{Q}_1(t_c)$. Hence, there is an embedding e from a pattern $P_1 \in \mathcal{Q}_1$ to t_c such that $e(out(P_1)) = o_c$. As both P_1 and P_2 are in $XP^{\{[\,],*\}}$ (i.e., they do not contain descendant edges), by definition of homomorphism we conclude that e constitutes a homomorphism from P_1 to P_2. Thus, Proposition 26 implies that $P_2 \sqsubseteq P_1$.

Complexity: In order to decide whether or not the containment $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ holds, we check for every pattern of $\mathcal{Q}_2$ if there is a homomorphism from a pattern of $\mathcal{Q}_1$. In the worst case, we check the existence of homomorphism for every possible combination of one pattern of $\mathcal{Q}_2$ and one pattern of $\mathcal{Q}_1$. Since we can decide whether there is homomorphism between two patterns in $XP^{\{//,*\}}$, in polynomial time to the sizes of the patterns [MS04], we conclude that deciding $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ is $PTIME$ w.r.t. the sizes of the sets. □

6.1.2 Deciding equivalence of patterns in $XP^{\{//,*\}}$

In this section, we give a necessary and sufficient condition for efficiently deciding equivalence of two pattern in $XP^{\{//,*\}}$. More specifically, we prove in the following proposition that the existence of a d-isomorphism suffices in order to decide such an equivalence.

Proposition 33. *Let two patterns P_1 and P_2 in $XP^{\{//,*\}}$. Then $P_2 \equiv P_1$* ***if and only if*** *$P_1 \sim_d P_2$. In this case, we decide equivalence in $PTIME$.*

Proof. **(If part)** Proof of this part immediately follows from Proposition 26 and definition of d-isomorphism.

(Only-If part) Let $P_2 \equiv P_1$. Then Proposition 29 implies that the depth d_1 of P_1 is equal to the depth d_2 of P_2, i.e. $d_1 = d_2$; and for every two nodes n_1, n_2 that appear in the same depth in P_1, P_2, respectively, we have that $label(n_1) = label(n_2)$. Hence, by definition of core pattern we have that $\widehat{P_1}$ and $\widehat{P_2}$ are isomorphic. In addition, considering that h is the isomorphism from $\widehat{P_1}$ to $\widehat{P_2}$, we conclude that for each core path $p_1 = u, \ldots, v$ of P_1 and the core path $p_2 = h(u), \ldots, h(v)$ of P_2, p_1 and p_2 have the same length. Moreover, by definition of d-homomorphism and Proposition 26, we conclude that h is a d-homomorphism from P_1 to P_2; which implies that for each core path $p_1 = u, \ldots, v$ of P_1 and the core path $p_2 = h(u), \ldots, h(v)$ of P_2, if p_1 has only child edges then p_2 also has only child edges. By symmetry, we also conclude that if p_2 has only child edges then p_1 also has only child edges. Thus, by definition of d-isomorphism, h is a d-isomorphism from P_1 to P_2.

Complexity: As deciding whether there is a d-homomorphism is $PTIME$, finding a d-isomorphism is also $PTIME$. □

Here, we have to notice that a significant remark is implied by the above result. Since the definition of a d-isomorphism does not specifies any requirement about the place and the number of descendant edges appearing in mapped core paths, we can easily see that either moving the descendant edges inside a core path or replacing the child edges with descendant edges in a core path with at least one descendant edge, the equivalence between patterns is not violated. This result is formally described by the following remark.

Remark 10. Let a pattern P such that there is a core path p in P with at least one descendant edge. If P' is the pattern obtained from P by replacing edges in p such that p has $k \geq 1$ descendant edges, then the following hold $P \equiv P'$.

Example 59. Consider the patterns P_1 and P_2 illustrated in Figure 6.3. Notice that there is a d-isomorphism from P_1 to P_2; which maps v_1, v_4, v_6 on u_1, u_4, u_6, respectively. Hence, $P_1 \equiv P_2$. In addition, we can see that in the first core path of P_1 there is only one descendant edge, instead of its image on P_2 which has two descendant edges appearing in different positions. In the second core path of P_1 notice that the existence of the d-isomorphism is not influenced by moving the descendant edge in its image. □

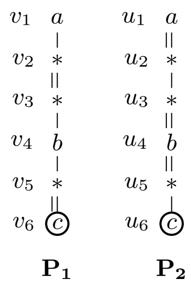


Figure 6.3: Descendant edges inside core paths; $P_1 \equiv P_2$

6.1.3 Containment of unions in the fragment of $XP^{\{//,*\}}$

Focusing, now, on the fragment of $XP^{\{//,*\}}$, we noticed (see Example 58) that deciding the containment between patterns included in two sets does not suffice in order to decide the containment of the sets. In this section we decide containment and equivalence of sets of patterns in $XP^{\{//,*\}}$ by translating the descendant edges of patterns into a bounded set of paths. Intuitively, using the recursive nature of the descendant edge, we replace each pattern included in the initial sets with an equivalent set of patterns in which each core path has either only child edges or one descendant edge and fixed length. Forcing descendant edges to appear only in core paths of specific length, the decision of the containment is given by finding d-homomorphisms in the direction that the initial sets specify.

Consider two sets $\mathcal{Q}_1$, $\mathcal{Q}_2$ of patterns in $XP^{\{//,*\}}$ such that $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$. Hence, for every tree t, each node contained in $\mathcal{Q}_2(t)$ is also contained in $\mathcal{Q}_1(t)$. Namely, if a path of a tree is accepted by $\mathcal{Q}_2$, then it is also accepted by $\mathcal{Q}_1$. A pattern, now, in $XP^{\{//,*\}}$ specify the form of the accepted paths by requiring a set of labels to appear in each of them (described by the non-wildcard labeled nodes of the pattern) on specific order, and also by specifying the distances between these labels. The order of the required labels between the root and the output is described by its core pattern. Hence, the containment of two sets implies that the core pattern of each pattern in $\mathcal{Q}_2$ is mapped, using a homomorphism, by a core pattern of a pattern in $\mathcal{Q}_1$.

Example 60. Continuing the Example 58, notice that the core pattern of P_4 is a path which is rooted by an a-node. Its end node is a c-node and it also has an internal node labeled by b. The core pattern, now, of P_1 specifies the same label requirements with that specified by core pattern of P_4 (notice that the core patterns of P_1 and P_4 are isomorphic). On the other hand, the core pattern of P_2 specifies more relaxed conditions on the labels it accepts than the labels required by the core pattern of P_4. More specifically, the core pattern of P_2 says that each path accepted by P_2 has root node labeled by a and end node labeled by c; which constitute a subset of the label requirement provided by P_4. This is formally given by the existence of a homomorphism from the core pattern of P_2 to the core pattern of P_4. □

The distances, now, of the images of the core nodes in each path accepted by a pattern are determined by the core paths of the pattern. More specifically, if a core path has length n and it has only child edges, then in every accepted path the images of its core endpoints have distance equal to n edges. Here, we notice that in the case that all patterns in both $\mathcal{Q}_1$ and $\mathcal{Q}_2$ have only child edges (i.e., they are in $XP^{\{*\}}$), the decision of containment is given by the Theorem 19.

On the other hand, the existence of a descendant edge in a core path captures a set of distances. Namely, if a core path has length n and it has at least one descendant edge, then the distance of the images of its core endpoinds is either equal to or greater than n edges. This feature implies that for a pattern P in $\mathcal{Q}_2$, there may be multiple patterns in $\mathcal{Q}_1$ that accept the paths accepted by P. An example of this case represented in Example 58; where the paths accepted by P_4 can be partitioned in two blocks such that P_1 accepts the paths of one block and P_2 accepts the paths of the other block. Hence, a pattern in $XP^{\{//,*\}}$ which has at least one descendant edge can be equivalently rewritten by a set of patterns also in $XP^{\{//,*\}}$; a property which is described by the concept of descendant unrolling [CW10] and it is proved by Proposition 34.

Definition 22. Let a pattern P in $XP^{\{//,[\,],*\}}$, (u, v) be an edge in $\mathcal{E}_{//}(P)$ and k be a positive integer. The k-**unrolling** of (u, v), denoted $Unroll^k_{(u,v)}(P)$, is the set of patterns $\{P_1, \ldots, P_{k+1}\}$, where

- the pattern P_1 is obtained from P by simply replacing the descendant edge (u, v) with a child edge;
- for each $i = 2, \ldots, k$, the pattern P_i is obtained from P by replacing (u, v) with the path $u, z_1, \ldots, z_{i-1}, v$ such that $z_1, \ldots, z_{i-1}$ are labeled by $*$ and all edges of the path are child edges; and

- the pattern P_{k+1} is obtained from P by replacing (u, v) with the path u, z_1, z_2, $\dots$, z_k, v such that z_1, z_2, $\dots$, z_k are labeled by $*$, the edge (z_k, v) is descendant edge and all the other edges of the path are child edges.

Remark 11. Notice, here, that if we replace the pattern P_{k+1} in the k-unrolling of (u, v) with the patterns of ℓ-unrolling of (z_{k+1}, v) appearing in P_{k+1} then we get the $Unroll^{(k+\ell)}_{(u,v)}(Q)$.

The following proposition shows that each descendant unrolling of a pattern is equivalent to this pattern.

Proposition 34. *[CW10][1] Let a pattern P in $XP^{\{//,[\,],*\}}$, (u, v) be an edge in $\mathcal{E}_{//}(P)$ and k be a positive integer. Then we have that the k-unrolling of (u, v) is equivalent to P.*

Proof. Let $\{P_1, \dots, P_{k+1}\}$ be the k-unrolling of (u, v). Consider an arbitrary embedding e from P to a tree t such that the descendant edge (u, v) maps on a path p of t whose length is $m \geq 1$. If $m \leq k$ then by definitions of descendant unrolling and embedding there is an embedding e' from P_m to t such that $o_e = o_{e'}$. On the other hand, if $m > k$ then by definitions of descendant unrolling and embedding there is an embedding e'' from P_{k+1} to t such that $o_e = o_{e''}$. Hence, $P \sqsubseteq Unroll^k_{(u,v)}(P)$. In addition, the construction of descendant unrolling of a pattern P implies that there is a homomorphism from P to each pattern of the unrolling; hence each pattern of $Unroll^k_{(u,v)}(P)$ is contained in P. Thus, we conclude $P \equiv Unroll^k_{(u,v)}(P)$. □

Example 61. Consider the pattern P_4 illustrated in Figure 6.2. Constructing the 1-unrolling of descendant edge (n_2, n_3), we firstly replace the edge (n_2, n_3) with child edge. The resulted pattern is given by the pattern Q_1; illustrated in Figure 6.4. Continuing, we replace (n_2, n_3) with the path n_2, z, n_3, where z is labeled by $*$, (z, n_3) is descendant edge and (n_2, z) is child edge. The pattern Q_2 illustrated in Figure 6.4 describes this substitution. Hence, the set $\{Q_1, Q_2\}$ constitutes the 1-unrolling of (n_2, n_3). In addition, Proposition 34 implies that $P_4 \equiv \{Q_1, Q_2\}$. □

Intuitively, unrolling descendant edges of a core path several times (using the transitive property described in Remark 11), we can construct sets

[1] This proposition is introduced in [CW10] where the authors consider bag semantics for the evaluation of a pattern on a tree (i.e., multiple occurrences of a node in the result are allowed) and each tree is enforced by an ordering over its nodes.

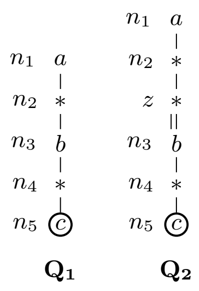


Figure 6.4: The 1-unrolling of n_1, n_2, n_3 of P_4, illustrated in Figure 6.2

of patterns which describe partitions of the paths accepted by the initial pattern. For example, the patterns Q_1, Q_2 illustrated in Figure 6.4 form a partition of the paths accepted by P_4 (where $Unroll^1_{(n_2,n_3)}(P_4) = \{Q_1, Q_2\}$) such that Q_1 accepts the paths in which the image of n_1 is exact two edges far from the image of n_3 and Q_2 the paths in which the image of n_1 is more than two edges far from the image of n_3. Moreover, Proposition 34 implies that there is no path which is accepted by one of the Q_1 or Q_2 and is not accepted by P_4.

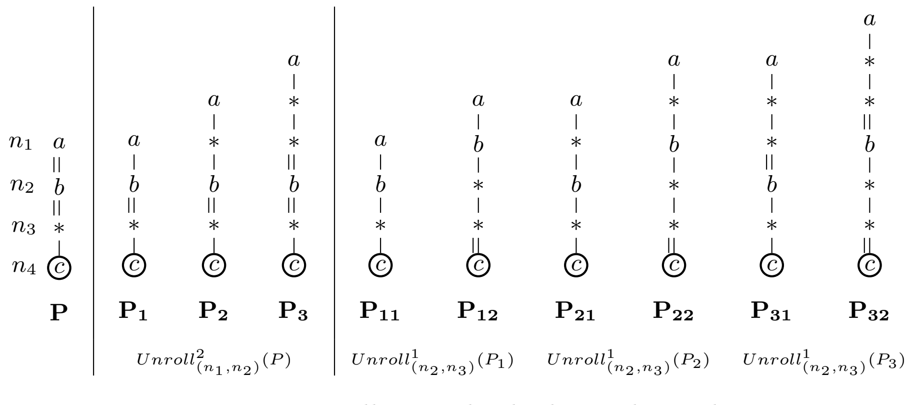
Using, now, the descendant unrolling of a pattern we can reveal cases in which a set of patterns is required for covering (all the paths accepted by) a single pattern. More specifically, although the single pattern is not contained in any pattern of the set, the set contains the pattern. This is revealed by appropriately partitioning the paths accepted by the single pattern such that for each block of the partition there is a pattern of the set that accepts the paths included in the block. Such a partitioning is achieved using descendant unrollings. Moreover, in order to decide whether or not a pattern of the set accepts (the paths included in) a certain block (which is described by a pattern included in an unrolling) we search for homomorphisms. An example of such a case was given by the Example 58. The following example continues the Example 58 and shows that unrolling a descendant edge we can decide containment of two sets of patterns by simply finding homomorphisms.

Example 62. Continuing the Example 58, notice that replacing the pattern P_4 in $\mathcal{Q}_2$ with the patterns in $Unroll^1_{(n_2,n_3)}(P_4)$ illustrated in Figure 6.4, an equivalent set is produced. Let $\mathcal{Q}'_2$ be this set. Hence, $\mathcal{Q}'_2 \equiv \mathcal{Q}_2$. Moreover, we can decide whether or not the containment $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ holds by equivalently deciding whether $\mathcal{Q}'_2 \sqsubseteq \mathcal{Q}_1$ holds. Considering, now, the mapping h from

P_2 to Q_2 such that the nodes of P_2 maps on the nodes of Q_2 that appear in the same depth, we conclude that h is a homomorphism. Moreover, we can easily verify that Q_1 and P_1 are isomorphic. In Example 58, we also showed that the pattern P_3 is contained in P_2 because of the existence of a homomorphism from P_2 to P_3. Consequently, each pattern in $\mathcal{Q}_2'$ is contained in a pattern of $\mathcal{Q}_1$; which formally shows that the containment $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ holds. □

A pattern, however, in $XP^{\{//,*\}}$ may contain multiple descendant edges appearing either inside the same core path or in different core paths. Hence, which of them we have to unroll and how many times are the two major problems that arise in order to decide the containment of two sets. The following example represents a case in which we have to unroll several times two descendant edges appearing in two different core paths in order to show that the containment holds.

Example 63. Let the sets of patterns $\mathcal{Q}_1 = \{P\}$ and $\mathcal{Q}_2 = \{P_{11}, P_{12}, P_{21}, P_{22}, P_{31}, P_{32}\}$, where all patterns are illustrated in Figure 6.5. It is easy to see that there is a homomorphism from P to each pattern of $\mathcal{Q}_2$. Hence, $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$. Notice, however, that there is not any homomorphism from a pattern of $\mathcal{Q}_2$ to P. If, now, we construct the 2-unrolling of the descendant edge (n_1, n_2) of P, then we get the set of patterns $\mathcal{Q}_1' = \{P_1, P_2, P_3\}$; which is equivalent to P. Further, unrolling the descendant edge (n_2, n_3) in each of the patterns P_1, P_2 and P_3 one time, and then replacing each pattern in $\mathcal{Q}_1'$ with its unrolling, we construct a set which is identical to $\mathcal{Q}_2$. Namely, this set is isomorphic to $\mathcal{Q}_2$. In addition, since this set is equivalent to $\mathcal{Q}_1$, we have that $\mathcal{Q}_1 \equiv \mathcal{Q}_2$. □



n_1
n_2
n_3
n_4
P
P1
P2
P3
P11
P12
P21
P22
P31
P32
$Unroll^2_{(n_1,n_2)}(P)$
$Unroll^1_{(n_2,n_3)}(P_1)$
$Unroll^1_{(n_2,n_3)}(P_2)$
$Unroll^1_{(n_2,n_3)}(P_3)$

Figure 6.5: Unrolling multiple descendant edges

In the above example we had to construct two different unrollings of two different descendant edges in order to form the appropriate partition of the paths accepted by P. However, notice that in the final set (i.e., the set constructed after the unrollings), all the descendant edges appear in core paths of length 3. Considering, now, a positive integer k, we refer to a core path which has a descendant edge and its length is less than k as *k-unrollable*. Instead, we say that a pattern in $XP^{\{//,*\}}$ is *k-unrolled* if it has not any k-unrollable core path [CW10]. Here, we give an algorithm which uses descendant unrollings in order to construct for a given pattern P in $XP^{\{//,*\}}$ and a positive integer k, a set of k-unrolled patterns which is equivalent to P.

Algorithm 1. (Construction of k-unroll-set)

- Input: a pattern P in $XP^{\{//,*\}}$.
- Output: The set U of k-unrolled patterns in $XP^{\{//,*\}}$.

1. Do the necessary substitutions of edges in P such that each core path which contains descendant edges is reformulated to a core path of which contains a unique descendant edge.
2. Set $U = \{P\}$.
3. For every pattern P' in U which contains a k-unrollable core path $p = u, \dots, v$ of length ℓ do the following:
    - Supposing that (z, v) is the unique descendant edge appearing in p, we replace P' in U with the patterns in $Unroll^{k-\ell}_{(z,v)}(P')$.

It easy to see that the above algorithm terminates for every pattern P in $XP^{\{//,*\}}$. We refer to the final set produced by the above algorithm as the **k-unroll-set** of P; and it is denoted as $Unroll^k(P)$. Moreover, since there may be multiple k-unrollable core paths in a pattern we suppose that the algorithm always chooses the topmost k-unrollable core path.

Example 64. Consider the pattern P illustrated in Figure 6.5. In Figure 6.5 we can also see the four iterations of the construction of $Unroll^3(P)$. In addition, it is easy to see that P has already the form that the Step 1 requires. Hence, we ignore this step.

Initially, now, we set $U = \{P\}$; illustrated in the first column. Then, replacing P with the 2-unrolling of the (n_1, n_2) we get the three patterns P_1, P_2 and P_3, which constitute the second instance of U (illustrated in second column). As we notice, none of them is 3-unrolled since in each of them the core path n_2, n_3, n_4 has length 2 and it also contains a descendant edge. Hence, we construct for each of them the 1-unrolling of (n_2, n_3). Finally, the $Unroll^3(P)$ contains the patterns illustrated in the third column (i.e., $Unroll^3(P) = \{P_{11}, P_{12}, P_{21}, P_{22}, P_{31}, P_{32}\}$). $\square$

Notice, here, that the construction of the unroll-set of a pattern P can be represented by a tree-like form; where each node of the tree is associated with an intermediate pattern produced during the construction of the unroll-set. More specifically, the pattern P is always placed on the root of the tree, and the relationship parent-children describes a descendant unrolling. Namely, the patterns included in the k-unrolling of the descendant edge of the intermediate pattern Q are the children of Q in this tree. Moreover, it is easy to see that the patterns appearing on leafs of the tree are the patterns included in the unroll-set. Using, now, the assumption in which we always choose the topmost descendant edge for unrolling, we can see that none of the top i core paths of each pattern appears in the level i (from the root) of the tree is k-unrollable. An example of such a representation is illustrated in Figure 6.6; which represents the construction of the 3-unroll-set of P illustrated in Figure 6.5.

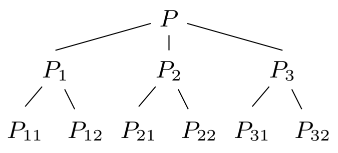


Figure 6.6: Tree-representation of an unroll-set

The above algorithm, now, guarantees that each unroll-set of a pattern P is equivalent to P. This feature follows from the fact that, in each step, the algorithm replaces a pattern in U with an equivalent a set of patterns. More specifically, Proposition 33 implies that the reformulation of P in Step 1 produces a pattern which is equivalent to P. In addition, in each iteration of Step 3, we replace a pattern with the patterns produced by its descendant unrolling; hence, in each iteration, the equivalence is preserved. This feature is formally described by the following corollary.

Corollary 7. *Let a pattern P in $XP^{\{//,*\}}$. Then for every positive integer k, we have that $Unroll^k(P) \equiv P$.*

Notice, here, that the Step 1 of the above algorithm reformulates the core paths with descendant edges in such a way that each new core path contains only one descendant edge. Using this step we avoid the appearance of redundant patterns in the unroll-set. In addition, we speed up the construction of the set of k-unrolled patterns, since the existence of more descendant edges implies more descendant unrollings for the construction of k-unrolled patterns. The following example describes these observations.

Example 65. Let the pattern P illustrated in Figure 6.7. Unrolling, now, 1 time the descendant edge (n_1, n_2) we get the two patterns illustrated in the second column of Figure 6.7; i.e., the patterns P_1 and P_2. It is easy to see that P_1 is not 3-unrolled. Unrolling, now, the descendant edge appearing in P_1 one more time, we get the patterns P_{11} and P_{12}; which are 3-unrolled. Unrolling, now, firstly the edge (n_2, n_3) two times and then the edge (n_1, n_2) one time, we get a similar final set; which also contains three patterns, two of which are equivalent.

Considering that the patterns in each of the three columns of Figure 6.7 constitute a set, we can easily conclude that these sets are equivalent. Moreover, if we ignore the Step 1, then these sets represent the instances of U in the Algorithm 1. On the other hand, running the Algorithm 1 in order to produce the 3-unroll-set of P, the pattern P_1 is obtained by reformulating P, as the Step 1 states. Then, a single unrolling of the descendant edge appearing in P_1 is required. The 3-unroll-set of P is given by the end of the first iteration of Step 3, and contains the patterns P_{11} and P_{12} (i.e., the pattern P_2 is not constructed by the algorithm). □

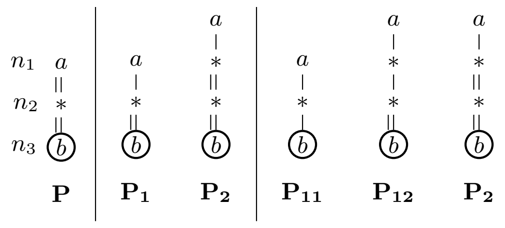


Figure 6.7: Multiple descendant edges in a core path

In the above example, notice that unrolling multiple descendant edges that appear in the same core path we need further computation in order to produce the k-unroll-set of the pattern. Moreover, in this case, at the end of each descendant unrolling, the produced set contains multiple equivalent patterns. Since, now, the above algorithm (using Step 1) eliminates multiple descendant edges from each core path, we achieve no equivalent patterns to appear in the k-unroll-set. This feature guarantees less overlapping of the results of the patterns appearing in the k-unroll-set.

Notice, here, that in a k-unrolled pattern each core path either has only child edges or its length is greater than or equal to k. Moreover, notice that as k approaches infinity the $Unroll^k(P)$ approaches an infinite set in which all descendant edges of P are replaced by every possible path that contains wildcard-labeled internal nodes; i.e., $Unroll^k(P)$ approaches an infinite set of patterns in $XP^{\{*\}}$ which is also equivalent to P. Since, now, we decide the containment of two sets of patterns in $XP^{\{*\}}$ by finding homomorphisms between the patterns included in two sets (Theorem 19), one may think of finding a positive integer k such that the problem of containment of two sets of patterns is reduced on finding homomorphisms between the patterns of their k-unroll-sets. As we can see in the following there exists such a number k, and it follows from the longest core path appearing in a pattern. The existence, however, of descendant edges in arbitrary positions inside the core paths require a relaxation of homomorphism; hence, similarly to the case of containment of single patterns, the d-homomorphism suffices.

Consider, now, a set $\mathcal{Q}$ of patterns in $XP^{\{//,*\}}$ and the length k_{max} of the longest core path appearing in the patterns of $\mathcal{Q}$. Constructing the $Unroll^{k_{max}}(\mathcal{Q})$ it is easy to see that the descendant edges appear only in the core paths of length k_{max}. However, there may be core paths of length k_{max} which do not have any descendant edge. On the other hand, if we construct the $Unroll^{k_{max}+1}(\mathcal{Q})$ we both force descendant edges to appear only in core paths of length $k_{max}+1$ and guarantee that every core path of such length has a descendant edge. All the other core paths has only child edges. This observation consists of the key for deciding containment of two sets of patterns in $XP^{\{//,*\}}$. The solution of the problem is formally given by the following lemma.

Lemma 4. *Let the sets $\mathcal{Q}_1$, $\mathcal{Q}_2$ of patterns in $XP^{\{//,*\}}$ and $k = k_{max}+1$ where k_{max} is the length of the longest core path appearing in a pattern in $\mathcal{Q}_1 \cup \mathcal{Q}_2$. Then $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$* ***if and only if*** *for each pattern $P_2 \in Unroll^k(\mathcal{Q}_2)$ there is a pattern $P_1 \in Unroll^k(\mathcal{Q}_1)$ such that there is a d-homomorphism from P_1 to P_2.*

Proof. **(If part)** The proof of this part immediately follows from Proposition 26 and the definition of the union of sets.

(Only-If part) Let $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ and an arbitrary pattern $P_2 \in Unroll^k(\mathcal{Q}_2)$. The Corollary 7 implies that $Unroll^k(\mathcal{Q}_2) \sqsubseteq Unroll^k(\mathcal{Q}_1)$. Constructing, now, the 1-canonical model of P_2 (denoted t for short), we conclude that $Unroll^k(\mathcal{Q}_2)(t) \subseteq Unroll^k(\mathcal{Q}_1)(t)$. Hence, there is a pattern $P_1 \in$

$Unroll^k(\mathcal{Q}_1)$ from which there is an embedding e to t such that $e(out(P_1)) = o^c_{P_2}$.

In addition, since $k = k_{max} + 1$, the core paths of lengths less than k appearing in the patterns in $Unroll^k(\mathcal{Q}_1) \cup Unroll^k(\mathcal{Q}_2)$, do not have any descendant edge (which is implied by the construction of unroll-set). Thus, for each core path p_1 of P_1 which contains only child edges, e maps p_1 on a path p_2 of t, where p_2 corresponds to a core path of P_2 which does not have any descendant edge. By the definitions of embedding and d-homomorphism we conclude that e constitutes a d-homomorphism from P_1 to P_2. $\square$

The condition, now, given by the previous lemma in order to decide the containment $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$ can be simplified by avoiding to construct the unroll-sets of both $\mathcal{Q}_1$ and $\mathcal{Q}_2$. More specifically, since the Proposition 34 and the construction of unroll-set imply that there is a homomorphism from a pattern to each pattern of its unroll-set, we can decide the above containment by finding d-homomorphisms directly from the patterns of $\mathcal{Q}_1$ to the patterns of the unroll-set of $\mathcal{Q}_2$. This conclusion is formally described by the following theorem.

Theorem 20. *Let the sets $\mathcal{Q}_1$, $\mathcal{Q}_2$ of patterns in $XP^{\{//,*\}}$ and $k = k_{max}+1$ where k_{max} is the length of the longest core path appearing in a pattern in $\mathcal{Q}_1 \cup \mathcal{Q}_2$. Then $\mathcal{Q}_2 \sqsubseteq \mathcal{Q}_1$* ***if and only if*** *for each pattern $P_2 \in Unroll^k(\mathcal{Q}_2)$ there is a pattern $P_1 \in \mathcal{Q}_1$ such that there is a d-homomorphism from P_1 to P_2.*

Deciding, now, whether two sets of patterns in $XP^{\{//,*\}}$ are equivalent or not, we construct their unroll-sets and we search for d-homomorphisms in both directions.

6.1.4 Equivalence of a pattern and a union of patterns in $XP^{\{//,*\}}$

In the case, now, of the equivalence of a pattern P and a set of patterns $\mathcal{Q}$ in $XP^{\{//,*\}}$, we can use the Theorem 20 in order to decide such a equivalence. Namely, we construct the unroll-sets of both P and $\mathcal{Q}$, and then we check for d-homomorphisms in both directions. The existence, however, of a single pattern in one set implies a tighter condition in order to decide equivalence. More specifically, the patterns in the unroll-set of P must be d-isomorphic to the patterns of a subset of the unroll-set of $\mathcal{Q}$. Moreover, an additional requirement is that each pattern in the unroll-set of $\mathcal{Q}$ have to be contained in P. In the following, we denote $\mathcal{Q}_1 \sim_d \mathcal{Q}_2$ if for each pattern in the set of patterns $\mathcal{Q}_1$ there is a d-isomorphic pattern in the set of patterns $\mathcal{Q}_2$, and vice versa.

Proposition 35. *Let a pattern P, and a set $\mathcal{Q}$ of patterns in $XP^{\{//,*\}}$. Also consider the positive integer $k = k_{max} + 1$, where k_{max} is the length of the longest core path appearing in a pattern of $\mathcal{Q} \cup \{P\}$. Then $P \equiv \mathcal{Q}$ **if and only if** both the following hold:*

1. *For each pattern Q in $\mathcal{Q}$, we have $Q \sqsubseteq P$; and*

2. *there is a subset U of $Unroll^k(\mathcal{Q})$ such that $Unroll^k(P) \sim_d U$.*

Proof. **(If part)** Proof of this part immediately follows from Proposition 26 and Theorem 20.

(Only-If part) We can easily see that the condition 1 is immediately follows from the definition of equivalence of sets.

In order to prove, now, the condition 1, Lemma 4 implies that to each pattern of $Unroll^k(P)$ there is a d-homomorphism from a pattern of $Unroll^k(\mathcal{Q})$, and vice versa. Considering an arbitrary pattern P_1 of $Unroll^k(P)$ we conclude that there is pattern P_2 in $Unroll^k(\mathcal{Q})$ from which there is a d-homomorphism to P_1. Similarly, there is a pattern P_3 also in $Unroll^k(P)$ from which there is a d-homomorphism to P_2. As P_1 and P_3 are contained in $Unroll^k(P)$ (i.e., they are produced from the same pattern), it is easy to verify that $\widehat{P_1} \sim \widehat{P_3}$. Similarly to the proof of Lemma 4, we conclude that $P_1 \sim P_2 \sim P_3$. Hence, there is a subset U of $Unroll^k(\mathcal{Q})$ such that $Unroll^k(P) \sim_d U$. □

This result shows that if a pattern P is equivalent to a set $\mathcal{Q}$ of patterns (all patterns are defined in $XP^{\{//,*\}}$) then there is a subset $\mathcal{Q}'$ of $\mathcal{Q}$ such that $P \equiv \mathcal{Q}' \equiv \mathcal{Q}$. This subset can be identified by checking which patterns of Q give patterns in their unroll-sets which are isomorphic to the patterns included in the corresponding unroll-set of P. Hence, identifying this subset, the conditions that each pattern in it satisfies are straightforwardly implied by the definitions of unroll-set and d-homomorphism, and are formally given by the following corollary.

Corollary 8. *Let a pattern P, and a set $\mathcal{Q}$ of patterns defined in $XP^{\{//,*\}}$ such that $P \equiv \mathcal{Q}$. Then there is a subset $\mathcal{Q}'$ of $\mathcal{Q}$ which is equivalent to both $\mathcal{Q}$ and P, and for each pattern Q in $\mathcal{Q}'$ the following hold.*

1. *$\widehat{P} \sim \widehat{Q}$; and*

2. *if h is the isomorphism from $\widehat{P}$ to $\widehat{Q}$ then h is a d-homomorphism from P to Q.*

Intuitively, the second condition shows that for each core path p of P the $h(p)$ is not longer than p, and if p has only child edges then $h(p)$ has only child edges and the same length with p. In addition, considering $k = k_{max} + 1$, where k_{max} is the length of the longest core path appearing in a pattern of $\mathcal{Q} \cup \{P\}$, it is easy to see that for each pattern in $Unroll^{k_{max}}(\mathcal{Q}')$ (resp. in $Unroll^{k_{max}}(P)$) there is an isomorphic pattern in $Unroll^{k_{max}}(P)$ (resp. in $Unroll^{k_{max}}(\mathcal{Q}')$).

6.2 Rewriting pattern using views

In this section, we focus on describing how a pattern in $XP^{\{//,*\}}$ can be equivalently rewritten using a set of views. More specifically, we will show that the union of single-view rewritings is required in order to equivalently rewrite a pattern in $XP^{\{//,*\}}$. In addition, we give an algorithm which finds an equivalent rewriting of a pattern using a set of views (if there exists any). We also show that this algorithm is sound but not complete. However, the algorithm guarantees that if there is any equivalent rewriting, then it outputs a rewriting and if there is not any, then it outputs nothing.

In the following example, we describe a case in which multiple single-view rewritings can be used to construct a set which is equivalent to an unroll-set of a single pattern.

Example 66. Let the pattern P and the set $\mathcal{V}$ of views, illustrated in in Figure 6.8. We can easily notice that there is no pattern R in $XP^{\{//,*\}}$ such that applying R on the result of a view in $\mathcal{V}$ we equivalently answer P. Namely, there is no R such that for a view V in $\mathcal{V}$ we have $R \circ V \equiv P$. However, for each view in $\mathcal{V}$ we can find at least one single-view contained rewriting. the patterns R_1, R_2 and R_3 illustrated also in Figure 6.8 represent such contained rewritings. More specifically, the patterns $R_1 \circ V_1$, $R_2 \circ V_2$, $R_3 \circ V_3$ and $R_3 \circ V_4$ are contained in P. These rewritings, however, have another significant property. The union of their results is always equal to the answer of P. In order to verify this we construct the 3-unroll-set of P. This unroll-set is illustrated in the bottom-right corner of Figure 6.8; and it is denoted as $\mathcal{P}$. Notice, now, that constructing the 3-unroll-set of $R_1 \circ V_1$ the only patterns included in this set are P_1 and P_4. Moreover, the 3-unroll-set of $R_2 \circ V_2$ and the 3-unroll-set of $R_3 \circ V_3$ are the sets $\{P_1, P_2, P_3\}$ and $\{P_2, P_5\}$, respectively. Finally, the pattern $R_3 \circ V_4$ is the same with P_6. It is easy to see, here, that all the above rewritings cover at least one distinct pattern of the 3-unroll-set of P; and all of them cover all the patterns included in the 3-unroll-set of P. Consequently, getting the union of the above single-view rewritings we equivalently rewrite P using the views in $\mathcal{V}$. □

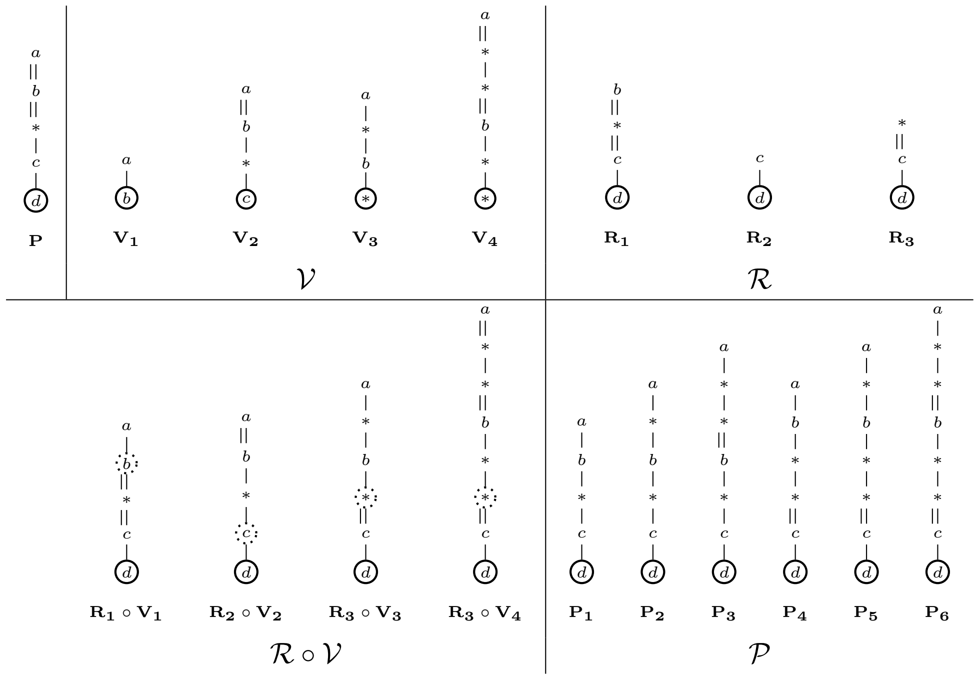


Figure 6.8: A union of single-view rewritings is required

Notice, here, that if for a given tree t we store the results of applying the above views on t, then practically the result $\mathcal{R} \circ \mathcal{V}(t)$ (where $\mathcal{R} \circ \mathcal{V}$ is illustrated in Figure 6.8) may not be equal to $P(t)$. More specifically, consider the case that a subtree t' of t is resulted by multiple views of $\mathcal{V}$. Materializing, now, the results of the views, multiple copies of t' are stored. Therefore, applying the rewritings on the copies of t' and then getting union of their results, the multiple copies of t' are treated as distinct trees. Hence, there might be more than one occurrences of a tree in the result of $\mathcal{R} \circ \mathcal{V}$ than in the result of P. In this work, however, we consider that either the results of the views are materialized as references to the initial tree (i.e., we store only references to subtrees of the tree; instead of fully extraction of the subtrees and their materialization), or we apply each view on the tree and then its rewriting on the fly (i.e., we actually apply the pattern given by the composition of the rewriting and the view). The latter case has significant practical interest for the optimization of complex XML queries (e.g.,XQuery expressions); in which multiple patterns are applied on a tree during their evaluation.

We extend, now, the rewriting problem for patterns and views in $XP^{\{//,*\}}$, in order to capture similar to the above cases. Hence, considering a pattern and a set of views we focus on deciding whether or not there exist patterns such that applying these patterns on the trees resulted by the views we get all the trees resulted by the given pattern. As we noticed, however, in the above example, these patterns do not necessarily apply on all trees resulted by every view. More specifically, in order to answer the given pattern, we may apply a single rewriting on different views but not necessarily on all views. Hence, a solution of this problem is to find a combination of rewritings and views such that the trees resulted by the union of the results of the rewriting always cover the answers resulted by the given pattern. Such a combination of rewritings and views is called union rewriting and is formally defined as follows.

Definition 23. Let a pattern in $XP^{\{//,*\}}$ and a set of views $\mathcal{V}$ also in $XP^{\{//,*\}}$. We say that a set $\mathcal{R}$ of patterns in $XP^{\{//,*\}}$ is a **union rewriting** of P using the views $\mathcal{V}$ if for each R in $\mathcal{R}$ there is a subset $\mathcal{V}_R$ of $\mathcal{V}$ such that for each tree t we have that $\bigcup_{R\in\mathcal{R}} R(\mathcal{V}_R(t)) = P(t)$.

Using the Proposition 32 we conclude that in this case the pattern P is equivalent to $\bigcup_{R\in\mathcal{R},V\in\mathcal{V}_R}\{R \circ V\}$. We denote, now, as $\mathcal{R} \circ \mathcal{V}$ the set of patterns $\bigcup_{R\in\mathcal{R},V\in\mathcal{V}_R}\{R \circ V\}$.

A pattern, now, may have infinite union rewritings using a given set of views. This conclusion is easily implied by the concept of descendant unrolling.

Example 67. Continuing the Example 66, we unroll the descendant edge appearing in pattern R_3 once. The patterns included in the produced set are the $R_{31} = */c/d$ and the $R_{32} = */*//c/d$. It is easy to see, now, that the set of patterns $\{R_1 \circ V_1,\ R_2 \circ V_2,\ R_{31} \circ V_3,\ R_{32} \circ V_3,\ R_3 \circ V_4\}$ is also equivalent to P. Hence, the set that contains the patterns R_1, R_2, R_3, R_{31} and R_{32} is also a union rewriting of P using $\mathcal{V}$. □

Intuitively, if there is a pattern in a union rewriting that has at least one descendant edge, then the following happens: constructing sets of patterns by unrolling multiple times each descendant edge of the pattern we can find different union rewritings. The existence of infinite union rewritings, in this case, is easily concluded. This observation lead us to extend the concept of potential rewriting. Hence, a set of patterns $\mathcal{R}$ is a *potential union rewriting* of a pattern P using a set of views $\mathcal{V}$ in $XP^{\{//,*\}}$ if the following condition holds: if there exists a union rewriting of P using $\mathcal{V}$, then $\mathcal{R}$ is also a union rewriting.

6.2.1 Solving the problem of the existence of a union rewriting

In this section we show that natural candidates of a given pattern do not suffice in order to find a potential union rewriting of the pattern using a given set of views. In addition, we give a superset of natural candidates using which a potential union rewriting can be constructed.

Consider a pattern P and a set of views $\mathcal{V}$ such that all patterns are in $XP^{\{//,*\}}$. Moreover, supposing that there is a union rewriting $\mathcal{R}$ of P using $\mathcal{V}$, we conclude that $\mathcal{R} \circ \mathcal{V} \equiv P$. Therefore, there is a subset of $\mathcal{R} \circ \mathcal{V}$ such that each pattern in this subset satisfies the conditions described in Corollary 8. Focusing, now, on these conditions we can conclude properties that each view has to fulfill in order to be useful for the construction of a union rewriting of P.

Consider an arbitrary view V in $\mathcal{V}$ and a pattern R in $\mathcal{R}$ such that $R \circ V \in \mathcal{R} \circ \mathcal{V}$. Hence, the first condition of Corollary 8 implies that $\widehat{R \circ V}$ and $\widehat{P}$ are isomorphic; from which it follows that $\widehat{V}$ is not longer than $\widehat{P}$. In addition, if d_V is the depth of $\widehat{V}$ then for each $i = 1, \ldots, (d_V - 1)$, the labels of the i-th core nodes of $\widehat{P}$ and $\widehat{V}$ are identical. For the output node of V, however, we have to notice that if it is labeled by $*$ then in $R \circ V$ this node is either intermediate node of a core path or it is replaced during the composition by a non-wildcard labeled node (the case that $R \circ V \sim V$ is an exception and is captured similarly to the case that the output is a non-wildcard labeled node). On the other hand, if the output node is labeled by a non-wildcard, then its label has to be identical to the label of the d_V-th core node of $\widehat{P}$.

The second condition of the Corollary 8 implies further requirements about core paths of useful views. More specifically, by definition of d-homomorphism we conclude that for each $i = 1, \ldots, (d_V - 1)$, the core path q_V that enters the i-th core node of V has to satisfy the following condition: if the i-th core path q_P of P (i.e., q_P enters the i-th core node of P) has only child edges then q_V and q_P are isomorphic; otherwise the length of q_V is at least equal to the length of q_P. For the deepest core path, now, of V we have two cases, one if the output node is labeled by $*$ and one otherwise. Considering that it is not labeled by $*$, the requirements that the deepest core path q'_V of V has to fulfill are the same with that of every other core path of V. However, if the output node of V is labeled by $*$ then the only condition that q'_V has to satisfy is the following: if the d_V-th core path q'_P of P has only child edges then q'_V also has only child edges, and it is not longer than q'_P. This condition follows from the fact that in $R \circ V$ this core path must have the same length with q'_P, where R is a pattern such that $R \circ V$ is included in a union rewriting of P.

The above conclusions about the requirements that a useful view has to fulfill, are captured by the concept of *homomorphic suffix* of a pattern; which is defined as follows.

Let a view V and a pattern P both in $XP^{\{//,*\}}$ such that $d_V \leq d_P$, where d_V and d_P are the depths of $\widehat{V}$ and $\widehat{P}$, respectively. Also, consider the non-negative integer k such that $P^{\leq k}$ is the pattern whose output is the d_V-th core node of P. Considering, now, the deepest paths q_P and q_V of V and $P^{\leq k}$, respectively, we construct the pattern V' as follows. If the start nodes of q_P and q_V have the same label, the end node of q_V (i.e., the $out(V)$) is labeled by $*$ and either

1. both q_P and q_V have only child edges and q_V is not longer than q_P, or

2. q_P has a descendant edge,

then V' is obtained from V by replacing q_V with q_P (the $out(V)$ is the end node of q_P). Otherwise V' is identical to V. In this case, we say that V is a k-**homomorphic suffix** of P if there is an isomorphism from $\widehat{P^{\leq k}}$ to $\widehat{V'}$ which is also a d-homomorphism from $P^{\leq k}$ to V'.

From the definition of homomorphic suffix and the Corollary 8 immediately follows the following corollary.

Corollary 9. *Let a pattern P in $XP^{\{//,*\}}$ and a set $\mathcal{V}$ of views in $XP^{\{//,*\}}$ such that there is a union rewriting $\mathcal{R}$ of P using $\mathcal{V}$. If a pattern $R \circ V$ in $\mathcal{R} \circ \mathcal{V}$ satisfies the following conditions*

1. *$\widehat{R \circ V} \sim \widehat{P}$, and*

2. *if h is the isomorphism from $\widehat{P}$ to $\widehat{R \circ V}$ then h is a d-homomorphism from P to $R \circ V$,*

then there is a non-negative integer k such that V is a k-homomorphic suffix of P.

Example 68. Let the pattern P and the set $\mathcal{V}$ of views illustrated in Figure 6.9. The view V_1 has core pattern which is isomorphic to $\widehat{P^{\leq 2}}$. However, the isomorphism from $\widehat{P^{\leq 2}}$ to $\widehat{V_1}$ is not a d-homomorphism from $P^{\leq 2}$ to V_1. Hence, V_1 is not a homomorphic suffix of P. For V_2 and V_3, now, replacing their unique core path with the unique core path of $P^{\leq 2}$, we get patterns which are isomorphic to $P^{\leq 2}$. Hence, these views are 2-homomorphic suffices of P. Checking for an isomorphism from $P^{\leq 4}$ (which is identical to P) to either V_4 or V_6, we conclude that both views are not homomorphic suffices of P. The view V_5, however, fulfills the requirements of homomorphic suffix. In order to verify this, we construct the view V_5' which is obtained by replacing the deepest core path of V_5 with the deepest core path of $P^{\leq 4}$. V_5', now, is isomorphic to $P^{\leq 4}$; which implies that V_5 is a 4-homomorphic suffix of P. □

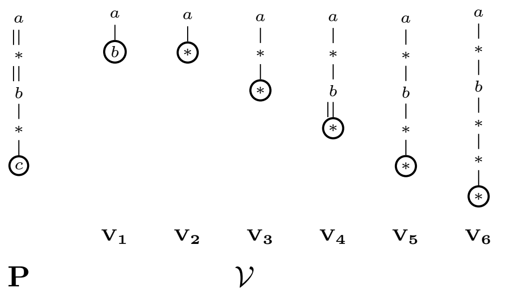


Figure 6.9: Homomorphic suffices of a pattern in $XP^{\{//,*\}}$

Notice, here, that for a view V which is not a homomorphic suffix of a pattern P, we cannot find a pattern R such that $R \circ V$ satisfies the conditions described in Corollary 8. For example, the view V_1 defined in Example 68 has core pattern which is identical to the 2-sub-pattern of P. However, for every pattern R, $R \circ V$ does not satisfy the second condition of the Corollary 8; since the mapping from the first two core nodes of P to the core nodes of V_1 violates the requirements of d-homomorphism. Intuitively, now, for every tree t, $V_1(t)$ contains elements labeled by b which are children of the a-labeled root element of t. However, in the paths of t that P accepts, a b-element is at least grandchild of the a-labeled root element of t. Similarly, the views V_4 and V_6 cannot be used in order to construct a set, similar to that described by Corollary 8.

In order, now, to find patterns such that applying them on the homomorphic suffices of a given pattern P, they give a union rewriting of P, we categorize the homomorphic suffices into the following categories. Considering a view V which is k-homomorphic suffix of P and q_V, q_P are the deepest core paths of V, $P^{\leq k}$, respectively, we say that V is a k-homomorphic suffix of

- **type 1**, if either $out(V)$ is labeled by $*$ and q_P has only child edges or V has a non-wildcard labeled output,
- **type 2**, if $out(V)$ is labeled by $*$ and q_P which has descendant edges, is longer than q_V,
- **type 3**, if $out(V)$ is labeled by $*$ and q_P which has descendant edges, is not longer than q_V.

Using, now, the above types of homomorphic suffices, we propose the following algorithm, which constructs a union rewriting (if there exists any) of a given pattern P using a given set of views $\mathcal{V}$. Moreover, in Theorem 21, we show that this algorithm constructs a potential union rewriting. In the following we use the notation $*//Q$ to denote the pattern that is obtained by adding a new root to the pattern Q which is labeled by $*$ and is connected with the former root with a descendant edge.

Algorithm 2.

- Input: a pattern P in $XP^{\{//,*\}}$ and a set $\mathcal{V}$ of views in $XP^{\{//,*\}}$.
- Output: the pair $(\mathcal{R}, \mathcal{R} \circ \mathcal{V})$, where $\mathcal{R}$ is a union rewriting of P using $\mathcal{V}$.

1. For each pattern k-homomorphic suffix V of P in $\mathcal{V}$ do the following: Let $m = k - i \cdot (\ell_P - \ell_V)$, where ℓ_P, ℓ_V are the lengths of the deepest core paths of $P^{\leq k}$, V, respectively, and $i = 1$ if $label(out(V)) = *$, otherwise $i = 0$.
   (a) If V is of type 1 then add the pattern $P^{\geq m} \circ V$ to $\mathcal{R} \circ \mathcal{V}$ and the pattern $P^{\geq m}$ to $\mathcal{R}$.
   (b) If V is of type 2 then add the pattern $P^{\geq m}_{//r} \circ V$ to $\mathcal{R} \circ \mathcal{V}$ and the pattern $P^{\geq m}_{//r}$ to $\mathcal{R}$.
   (c) If V is of type 3 then add the patterns $P^{\geq k} \circ V$, $*//P^{\geq k} \circ V$ to $\mathcal{R} \circ \mathcal{V}$ and the patterns $P^{\geq k}$, $*//P^{\geq k}$ to $\mathcal{R}$.
2. Check whether or not $\mathcal{R} \circ \mathcal{V}$ is equivalent to P; and if not return the pair of empty sets $(\emptyset, \emptyset)$. Otherwise return the $(\mathcal{R}, \mathcal{R} \circ \mathcal{V})$.

Example 69. Continuing the Example 68, we notice that V_5, V_2 and V_3 are of types 1, 2 and 3, respectively. The algorithm constructs the natural candidates $P^{\geq 1}_{//r}$ and $P^{\geq 4}$ and associates them to the views V_2 and V_5, respectively (i.e., since $P^{\geq 1}_{//r} \circ V_2$ and $P^{\geq 4} \circ V_5$ are added in $\mathcal{R} \circ \mathcal{V}$, we conclude that the patterns $P^{\geq 1}_{//r}$ and $P^{\geq 4}$ are applied on the results of views V_2 and V_5, respectively). For the view V_3, now, the algorithm constructs two patterns; which are the $P^{\geq 2}$ and $*//P^{\geq 2}$. In this case, we can easily see that the view V_3, alone, gives a union rewriting of P (i.e., $P^{\geq 2} \circ V_3 \cup (*//P^{\geq 2}) \circ V_3 \equiv P$). The same result holds for the view V_2 (notice that $P^{\geq 1}_{//r} \circ V_2$ is isomorphic to P). Moreover, the union $(*//P^{\geq 2} \circ V_3) \cup (P^{\geq 4} \circ V_5)$ is also equivalent to P. Consequently, the set $\mathcal{R} = \{P^{\geq 1}_{//r}, P^{\geq 2}, *//P^{\geq 2}, P^{\geq 4}\}$ which is produced by the above algorithm, is a union rewriting of P using the views V_2, V_3 and V_5. □

Theorem 21. *For a given pattern P in $XP^{\{//,*\}}$ and a set $\mathcal{V}$ of views in $XP^{\{//,*\}}$, the Algorithm 2 outputs a potential union rewriting of P using $\mathcal{V}$.*

Proof. By the Step 2 of Algorithm 2, we conclude that the output $\mathcal{R}$ of the algorithm is a union rewriting of P using $\mathcal{V}$; i.e., $\mathcal{R} \circ \mathcal{V} \equiv P$. We will prove, now, that $\mathcal{R}$ is a potential union rewriting of P using $\mathcal{V}$. More specifically, considering that there is a union rewriting $\mathcal{R}'$ of P using $\mathcal{V}$, we will show that the above algorithm constructs in Step 1 a set of patterns $\mathcal{R}_a$ such that $\mathcal{R}_a \circ \mathcal{V} \equiv P$. This is proved in the following two steps:

1. We show that there is a subset U of $\mathcal{R}' \circ \mathcal{V}$ such that $U \equiv \mathcal{R}' \circ \mathcal{V}$ and for each pattern $R \circ V$ in U there exists a pattern in $\mathcal{R}_a \circ \mathcal{V}$ which contains $R \circ V$.

2. Then we prove that $\mathcal{R}_a \circ \mathcal{V}$ is equivalent to $\mathcal{R}' \circ \mathcal{V}$ (i.e., equivalent to P) by also showing that each pattern in $\mathcal{R}_a \circ \mathcal{V}$ is contained by P.

By the Corollaries 8 and 9 we conclude that there is a subset U of $\mathcal{R}' \circ \mathcal{V}$ such that $U \equiv \mathcal{R}' \circ \mathcal{V}$ and for each pattern $R \circ V$ in U we have that V is a homomorphic suffix of P. Let $R \circ V$ be an arbitrary pattern in U, V is a k-homomorphic suffix of P, and $m = \ell_P - \ell_V$, where ℓ_P, ℓ_V are the lengths of the deepest core paths of $P^{\leq k}$, V, respectively. By Corollary 8 we have that $\widehat{R \circ V} \sim \widehat{P}$ and if h is the isomorphism from $\widehat{P}$ to $\widehat{R \circ V}$ then h is a d-homomorphism from P to $R \circ V$. Consider, now, that V has d_V core paths and R has d_R core paths.

Firstly, suppose that the output of V is labeled by a non-wildcard (i.e., V is of type 1). The core pattern of $P^{\geq m} \circ V$ is by definition isomorphic to

$\widehat{P}$; hence it is isomorphic to $\widehat{R \circ V}$ (notice that in this case $m = 0$). Let h_1 this isomorphism from $P^{\geq m} \circ V$ to $\widehat{R \circ V}$. We conclude, here, that the first d_V core paths of $P^{\geq m} \circ V$ and $\widehat{R \circ V}$ are isomorphic. Moreover, for every other core path p of $P^{\geq m} \circ V$ we have that $h(p) = h_1(p)$. Thus, h_1 is a d-homomorphism from $P^{\geq m} \circ V$ to $R \circ V$.

On the other hand, suppose that $out(V)$ is labeled by $*$. If, now, V is of type 1 then there is also an isomorphism h_2 from the core pattern of $P^{\geq m} \circ V$ to $\widehat{R \circ V}$. Similarly to the above case, we have that for each core path of $P^{\geq m} \circ V$, h_2 satisfies the requirements of d-homomorphism. On the other hand, if the d_V-th core path of P has a descendant edge we have two cases; one in which V is of type 2 and one in which V is of type 3. In the first case, the length of the d_V-th core path of $R \circ V$ is at least equal to the length of the d_V-th core path of P. Moreover, the d_V-th core path of $P^{\geq m}_{//r} \circ V$ is as longer as the d_V-th core path of P, and it also has at least one descendant edge. Hence, there is a d-homomorphism from $P^{\geq m}_{//r} \circ V$ to $R \circ V$. In the case that V is of type 3, the d_V-th core path of $R \circ V$ is longer than the d_V-th core path of P and at least as longer as the d_V-th core path of V. If, now, it is as longer as the d_V-th core path of V then there is a d-homomorphism from $P^{\geq k} \circ V$ to $R \circ V$. In any other case, there is a d-homomorphism from $*//P^{\geq k} \circ V$ to $R \circ V$.

Thus, for each case we proved that there is a pattern in $\mathcal{R}_a \circ \mathcal{V}$ which contains $R \circ V$. In addition, by definition $\mathcal{R}_a \circ \mathcal{V}$ we conclude that each pattern in it, is contained in P (because of the existence of d-homomorphisms). Consequently, $\mathcal{R}_a \circ \mathcal{V} \equiv P$ and $\mathcal{R}_a$ is a potential union rewriting of P using $\mathcal{V}$. $\square$

Here, we have to notice that the above algorithm is not always outputs the minimum union rewriting $\mathcal{R}$. Moreover, since the rewriting components constructed in polynomial time, its inefficient step is that of checking the equivalence between a single pattern and a union of patterns. In its output, now, the algorithm may contain patterns which are not natural candidates of the input pattern. This is implied by the existence of the type 3 homomorphic suffices in the input set of views. Hence, the natural candidates of a pattern in $XP^{\{//,*\}}$ do not suffice in order to find a potential union rewriting of the pattern using a set of views.

6.3 Conclusions and Future Work

In this chapter, we have studied the problem of rewriting patterns using multiple views, considering patterns which contain wildcards and descendant edges. More specifically, we proposed an algorithm which decides whether or not there exists any rewriting of a given pattern using a set of views, and outputs such a rewriting (if there exists any). This algorithm was based on equivalence between a single pattern and a set (or union) of patterns. The equivalence and containment problems for unions of patterns were also investigated. Especially, we proved necessary and sufficient conditions for both containment and equivalence in the cases of unions of $XP^{\{[\,],*\}}$ and $XP^{\{//,*\}}$ patterns. Moreover, we showed that the existence of a union rewriting may be required in order to equivalently rewrite a given pattern using a given set of views.

For future work, now, many problems arise. The extension of the union rewritings to the fragment of $XP^{\{//,[\,],*\}}$ is an open problem. We expect that most of the results of this chapter can be extended to this fragment. Although the containment problem for unions of $XP^{\{//,[\,],*\}}$ patterns has been solved [MS04], we expect that the further investigation of the equivalence problem for unions of such patterns may be helpful in order to solve the problem of finding union rewritings.

In addition, as multiple patterns appear in more complex XML queries (such as XQuery expressions), optimization techniques of patterns are useful. Therefore, another interesting problem is that of view selection. More specifically, for a given set of constraints over the sizes of views and the rewritings of a given set of patterns, we can search for optimal query plans that exploit possible sharing opportunities of the given patterns (similar problem to that investigated in Chapter 4).

Chapter 7

Conclusions

In this dissertation, we investigated high level query optimization techniques for both relational and XML databases. In particular, we focused on three essential problems appearing on query optimization setting; which are the query containment/equivalence problem, the query rewriting problem and the view selection problem. These problems were investigated for several major subclasses of the real-time queries (i.e., for subclasses of CQs, and subfragments of XPath queries).

On solving the view selection problem in relational databases, we searched for viewsets that give optimal equivalent rewritings for each query in the given workload, and satisfy a given storage constraint. In this perspective, we considered both bag and bag-set semantics to approximate the SQL semantics, and focused on finding at least one optimal solution for the given problem input (instead of finding every optimal solution). In order, now, to find such solutions we described the forms of each useful viewset (i.e., viewset which give equivalent rewritings for the given queries) and each CQ equivalent rewriting of a given CQ. Using this analysis we proposed an algorithm (called LGG-VSB) for the case of bag semantics, and a simple modification of it in order to solve the bag-set-oriented version of the problem. Moreover, we investigated cases in which optimal viewsets can be found using subexpressions of the given queries. For these special cases, we gave heuristics for the above algorithm.

The (equivalent) rewriting problem was also investigated for XML databases. In particular, focusing on XPath queries, we used the union operator to introduce rewritings of XPath queries using multiple views. In this perspective, we showed that there are cases in which the union operator is required in order to find an equivalent rewriting of a given query and set of views. Moreover, we described the form of useful views (i.e., views which give rewritings of XPath queries), considering a significant fragment of XPath. Using these results, we proposed an algorithm which outputs a potential union rewriting of a given query and set of views.

The techniques used on equivalently rewriting either relational or XML queries, were based on a detailed analysis of the problems of query containment and equivalence. In particular, we investigated the problems of bag and bag-set containment through a detailed analysis of special cases. In each of these cases, we computed the complexity and gave necessary and sufficient conditions. Deciding, however, CQ containment either under bag or under bag-set semantics remains an open problem. On the other hand, for XPath queries, we investigated in depth the problem of containment and equivalence of unions of patterns, considering that the patterns belong to one of the three major fragments of XPath.

For future work, now, the extension of the above problems to more general classes of queries is essential for many database research areas. For the bag and bag-set semantics, the problem of CQ containment remains open. Moreover, finding superclasses of CQs for which the problems of bag and bag-set containment are decidable, are interesting problems. The equivalence of superclasses of CQs for these semantics is also a problem that has not investigated in depth.

In addition, future work is needed on rewriting techniques that can perform correctly and efficiently for "real" SQL views on "real" SQL queries, including queries with aggregates, nesting, bag and bag-set semantics, and user-defined functions. Moreover, the conceptual framework of rewriting queries using views applies to problem in graph querying, Web service composition, and rewriting of XQuery. For the last one, the results of Chapter 6 contribute an interesting perspective for investigating efficient rewritings of XQuery expressions. Furthermore, due to the relevance of the rewriting problem and the information integration setting, the investigation of the data integration (especially, the LAV approach) considering either bag or bag-set semantics is also an open problem.

Bibliography

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: from relations to semistructured data and XML*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[AC05] Foto N. Afrati and Rada Chirkova. Selecting and using views to compute aggregate queries (extended abstract). In *ICDT*, pages 383–397, 2005.

[ACG$^+$09] Foto N. Afrati, Rada Chirkova, Manolis Gergatsoulis, Benny Kimelfeld, Vassia Pavlaki, and Yehoshua Sagiv. On rewriting xpath queries using views. In *EDBT*, pages 168–179, 2009.

[ACGP07] Foto Afrati, Rada Chirkova, Manolis Gergatsoulis, and Vassia Pavlaki. View selection for real conjunctive queries. *Acta Inf.*, 44(5):289–321, 2007.

[AD98] Serge Abiteboul and Oliver M. Duschka. Complexity of answering queries using materialized views. In *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 254–263, New York, NY, USA, 1998. ACM.

[ADGa] Foto N. Afrati, Matthew G. Damigos, and Manolis Gergatsoulis. On solving efficiently the view selection problem under bag and bag-set semantics. *Under Preparation.*

[ADGb] Foto N. Afrati, Matthew G. Damigos, and Manolis Gergatsoulis. Union rewritings for xpath fragments. *Under Preparation.*

[ADG08] Foto N. Afrati, Matthew G. Damigos, and Manolis Gergatsoulis. On solving efficiently the view selection problem under bag-semantics. In *2nd International Workshop on Business Intelligence for the Real Time Enterprise (BIRTE 2008)*, 2008.

[ADG09] Foto N. Afrati, Matthew G. Damigos, and Manolis Gergatsoulis. Query containment under bag and bag-set semantics. *submitted to Information Processing Letters (IPL)*, 2009.

[AE04] George E. Andrews and Kimmo Eriksson. *Integer Partitions*. Cambridge University Press, 2004.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[ALM02] Foto N. Afrati, Chen Li, and Prasenjit Mitra. Answering queries using views with arithmetic comparisons. In *PODS*, pages 209–220, 2002.

[ALU07] Foto N. Afrati, Chen Li, and Jeffrey D. Ullman. Using views to generate efficient evaluation plans for queries. *Journal of Computer and System Sciences*, 73(5):703 – 724, 2007.

[AYCLS01] Sihem Amer-Yahia, SungRan Cho, Laks V. S. Lakshmanan, and Divesh Srivastava. Minimization of tree pattern queries. In *SIGMOD '01: Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 497–508, New York, NY, USA, 2001. ACM.

[BG95] Francesco Bergadano and Daniele Gunetti. *Inductive Logic Programming: From Machine Learning to Software Engineering*. MIT Press, Cambridge, MA, USA, 1995.

[BÖB$^+$04] Andrey Balmin, Fatma Özcan, Kevin S. Beyer, Roberta Cochrane, and Hamid Pirahesh. A framework for using materialized xpath views in xml query processing. In *VLDB*, pages 60–71, 2004.

[BPT97] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized views selection in a multidimensional database. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 156–165, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[BR91] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. *The Journal of Logic Programming*, 10(3-4):255 – 299, 1991.

[CDO08] Bogdan Cautis, Alin Deutsch, and Nicola Onose. Xpath rewriting using multiple views: Achieving completeness and efficiency. In *WebDB*, 2008.

[CDOV09] Bogdan Cautis, Alin Deutsch, Nicola Onose, and Vasilis Vassalos. Efficient rewriting of xpath queries using query set specifications. *PVLDB*, 2(1):301–312, 2009.

[CFMS95] Silvana Castano, Maria Grazia Fugini, Giancarlo Martella, and Pierangela Samarati. *Database Security.* Addison-Wesley & ACM Press, 1995.

[CG00] Rada Chirkova and Michael R. Genesereth. Linearly bounded reformulations of conjunctive databases. In *Computational Logic*, pages 987–1001, 2000.

[CGLV00] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. Answering regular path queries using views. In *ICDE*, pages 389–398, 2000.

[CHS02] Rada Chirkova, Alon Y. Halevy, and Dan Suciu. A formal perspective on the view selection problem. *The VLDB Journal*, 11(3):216–237, 2002.

[CL03] Rada Chirkova and Chen Li. Materializing views with minimal size to answer queries. In *PODS*, pages 38–48, 2003.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

[CNS03] Sara Cohen, Werner Nutt, and Yehoshua Sagiv. Containment of aggregate queries. In *ICDT*, pages 111–125, 2003.

[Coh05] Sara Cohen. Containment of aggregate queries. *SIGMOD Rec.*, 34(1):77–85, 2005.

[CR02] Li Chen and Elke A. Rundensteiner. Xcache: Xquery-based caching system. In *WebDB*, pages 31–36, 2002.

[CV93] Surajit Chaudhuri and Moshe Y. Vardi. Optimization of real conjunctive queries. In *PODS '93: Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 59–70, 1993.

[CW10] Sara Cohen and Yaacov Y. Weiss. Bag equivalence of xpath queries. In *ICDT*, 2010.

[DT01] Alin Deutsch and Val Tannen. Containment and integrity constraints for xpath fragments. In *In KRDB*, 2001.

[DT03] Alin Deutsch and Val Tannen. Reformulation of xml queries and constraints. In *ICDT*, pages 225–241, 2003.

[EM03] Johann Eder and Michele Missikoff, editors. *Advanced Information Systems Engineering, 15th International Conference, CAiSE 2003, Klagenfurt, Austria, June 16-18, 2003, Proceedings*, volume 2681 of *Lecture Notes in Computer Science*. Springer, 2003.

[FLSY99] Daniela Florescu, Alon Y. Levy, Dan Suciu, and Khaled Yagoub. Optimization of run-time management of data intensive web-sites. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pages 627–638, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[GHRU97] Himanshu Gupta, Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Index selection for OLAP. In *ICDE '97: Proceedings of the Thirteenth International Conference on Data Engineering*, pages 208–219, Washington, DC, USA, 1997. IEEE Computer Society.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[GM05] Himanshu Gupta and Inderpal Singh Mumick. Selection of views to materialize in a data warehouse. *IEEE Trans. Knowl. Data Eng.*, 17(1):24–43, 2005.

[GMUW08] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book.* Prentice Hall Press, Upper Saddle River, NJ, USA, 2008.

[GT03] Gösta Grahne and Alex Thomo. Query containment and rewriting using views for regular path queries under constraints. In *PODS*, pages 111–122, 2003.

[Hal01] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[HRU96] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. Implementing data cubes efficiently. *SIGMOD Rec.*, 25(2):205–216, 1996.

[IR95] Yannis E. Ioannidis and Raghu Ramakrishnan. Containment of conjunctive queries: Beyond relations as sets. *ACM Trans. Database Syst.*, 20(3):288–324, 1995.

[JK84] David S. Johnson and Anthony C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1):167–189, 1984.

[JKV06] T. S. Jayram, Phokion G. Kolaitis, and Erik Vee. The containment problem for real conjunctive queries with inequalities. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 80–89, 2006.

[Klu88] Anthony C. Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1):146–160, 1988.

[KM99] Howard Karloff and Milena Mihail. On the complexity of the view-selection problem. In *PODS '99: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 167–173. ACM Press, 1999.

[KS08] Benny Kimelfeld and Yehoshua Sagiv. Revisiting redundancy and minimization in an xpath fragment. In *EDBT*, pages 61–72, 2008.

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 251–262. Morgan Kaufmann, 1996.

[LWZ06] Laks V. S. Lakshmanan, Hui Wang, and Zheng (Jessica) Zhao. Answering tree pattern queries using views. In *VLDB*, pages 571–582, 2006.

[MB06] Jim Melton and Stephen Buxton. *Querying XML,: XQuery, XPath, and SQL/XML in context (The Morgan Kaufmann Series in Data Management Systems) (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[MS99] Tova Milo and Dan Suciu. Type inference for queries on semistructured data. In *PODS '99: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 215–226, New York, NY, USA, 1999. ACM.

[MS04] Gerome Miklau and Dan Suciu. Containment and equivalence for a fragment of xpath. *J. ACM*, 51(1):2–45, 2004.

[MS05] Bhushan Mandhani and Dan Suciu. Query caching and view selection for xml databases. In *VLDB*, pages 469–480, 2005.

[NS03] Frank Neven and Thomas Schwentick. Xpath containment in the presence of disjunction, dtds, and variables. In *In ICDT*, pages 315–329, 2003.

[PH01] Rachel Pottinger and Alon Halevy. Minicon: A scalable algorithm for answering queries using views. *The VLDB Journal*, 10(2-3):182–198, 2001.

[Plo70] G.D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

[PS88] Jooseok Park and Arie Segev. Using common subexpressions to optimize multiple queries. In *ICDE*, pages 311–319, 1988.

[PV99] Yannis Papakonstantinou and Vasilis Vassalos. Query rewriting for semistructured data. In *SIGMOD Conference*, pages 455–466, 1999.

[RG02] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill Science/Engineering/Math, 3 edition, 2002.

[RSSB00] Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Efficient and extensible algorithms for multi query optimization. In *SIGMOD Conference*, pages 249–260, 2000.

[Sel88] Timos K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13(1):23–52, 1988.

[SY80] Yehoshua Sagiv and Mihalis Yannakakis. Equivalences among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.

[TS97] Dimitri Theodoratos and Timos K. Sellis. Data warehouse configuration. In *VLDB*, pages 126–135, 1997.

[TX04] Dimitri Theodoratos and Wugang Xu. Constructing search spaces for materialized view selection. In *DOLAP*, pages 112–121, 2004.

[TYÖ$^+$08] Nan Tang, Jeffrey Xu Yu, M. Tamer Özsu, Byron Choi, and Kam-Fai Wong. Multiple materialized view selection for xpath query rewriting. In *ICDE*, pages 873–882, 2008.

[Ull88] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II.* Computer Science Press, 1989.

[vdM97] Ron van der Meyden. The complexity of querying indefinite data about linearly ordered domains. *J. Comput. Syst. Sci.*, 54(1):113–135, 1997.

[Woo01] Peter T. Wood. Minimising simple xpath expressions. In *WebDB*, pages 13–18, 2001.

[Woo03] Peter T. Wood. Containment for xpath fragments under dtd constraints. In *ICDT*, pages 297–311, 2003.

[WY08] Junhu Wang and Jeffrey Xu Yu. Xpath rewriting using multiple views. In *DEXA*, pages 493–507, 2008.

[XO05] Wanhong Xu and Z. Meral Özsoyoglu. Rewriting xpath queries using materialized views. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 121–132. VLDB Endowment, 2005.

[XTZ06] Wugang Xu, Dimitri Theodoratos, and Calisto Zuzarte. Computing closest common subexpressions for view selection problems. In *DOLAP*, pages 75–82, 2006.

[Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Very Large Data Bases, 7th International Conference, September 9-11, 1981, Cannes, France, Proceedings*, pages 82–94. IEEE Computer Society, 1981.

[YCGL04] Jeffrey Xu Yu, Chi-Hon Choi, Gang Gou, and Hongjun Lu. Selecting views with maintenance cost constraints: Issues, heuristics and performance. *Journal of Research and Practice in Information Technology*, 36(2):89–110, 2004.

[YLH03] Liang Huai Yang, Mong-Li Lee, and Wynne Hsu. Efficient mining of xml query patterns for caching. In *VLDB*, pages 69–80, 2003.

[ZLFL07] Jingren Zhou, Per-Ake Larson, Johann Christoph Freytag, and Wolfgang Lehner. Efficient exploitation of similar subexpressions for query processing. In *SIGMOD Conference*, pages 533–544, 2007.

[ZLWL09] Rui Zhou, Chengfei Liu, Junhu Wang, and Jianxin Li. Containment between unions of xpath queries. In *DASFAA '09: Proceedings of the 14th International Conference on Database Systems for Advanced Applications*, pages 405–420, Berlin, Heidelberg, 2009. Springer-Verlag.

[ZS98] Antoine Zoghbi and Ivan Stojmenović. Fast algorithms for generating integer partitions. *Int. J. Comput. Math.*, 70(2):319–332, 1998.