PARTIAL ANSWERS IN INFORMATION INTEGRATION SYSTEMS : THEIR MEANING AND COMPUTATION

Victoria Kiricenko

A thesis
in
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Computer Science
Concordia University
Montréal, Québec, Canada

September 2003

Your file Votre référence
ISBN: 0-612-83908-7
Our file Notre référence
ISBN: 0-612-83908-7

Abstract

Partial Answers in Information Integration Systems : Their Meaning and Computation

Victoria Kiricenko

Information integration systems provide uniform interfaces to varieties of heterogeneous information sources. Our work focuses on query answering in such system. The current generation of query answering algorithms in local-as-view (source-centric) information integration systems all produce what has been thought of as "the best obtainable" answer, given the circumstances that the source-centric approach introduces incomplete information into the virtual global relations. However, this "best obtainable" answer does not include all information that can be extracted form the sources because it does not allow partial information.

We define the semantics of partial facts and introduce the notion of exact answer - that is the answer that includes partial facts. We also present two methods for computing exact answer, in such way that semantics of queries remain compositional. The first method is tableau-based and is a generalization of the "inverse-rules" approach. The second, much more efficient method, is a generalization of the rewriting approach, and is based on partial containment mappings introduced in the thesis. Furthermore, we provide two query rewriting algorithms that can be used to compute exact answer. Finally, we present experimental results confirming that computation of the exact answer can be done efficiently for all practical situations including the large-scale systems.

Acknowledgments

I would like to express the deepest gratitude to my supervisor Dr. Gösta Grahne for his guidance, enthusiasm and encouragement. Without his intellectual, moral, and financial support and persistent help this thesis would not have been possible. I acknowledge a debt, an appreciation, that extends beyond any words at my command.

I would also like to thank Dr. T. Radhakrishnan for helping me to make a decision to enroll in the Master Program and for giving me a good start in this program. My deepest appreciation to Dr. N. Shiri, Dr. V. Haarslev, and all other members of our database research group for creating remarkable research atmosphere and for providing discussions, ideas and suggestions essential to the planning and execution of this research. I would like to thank Dr. J. Opatrny and Dr. R. Shingal for contributing to my teaching career and for setting an example of what a real teacher should be. I am grateful to all faculty members and staff of the Department of Computer Science at Concordia University for contributing to my education, providing me an excellent work environment and making my studies and work at Concordia very enjoyable.

Many thanks to Jianfei Zhu, Alex Thomo, Ritesh Mukherjee and all of my other fellow students and friends for always being there for me. They have helped me more times than I can count.

The last but certainly not the least, I am forever indebted to my family for their inspiring love, encouragement, and support.

Contents

List of Figures

Chapter 1

Introduction

1.1 Information integration

Integrating heterogeneous data sources is a fundamental problem in databases, which has been studied extensively in the last two decades both from a formal and from a practical point of view (see e. g. [Ull97, Hal01, Len02]). Recently, mostly driven by the need to integrate data sources on the Web, much of the research on integration has focused on so called information integration systems. Information integration systems aim to provide a uniform query interface to multiple heterogeneous sources. The goal of an information integration system is to free the user from having to locate the sources relevant to a query, interact with each source in isolation, and manually combine data from multiple sources.

Examples of information integration applications include enterprise integration, querying multiple sources on the World Wide Web, and integration of data from distributed scientific experiments, just to name a few. The information sources in such an application may be traditional databases, legacy systems, or even structured files. To deal with the fact that different sources use different data models, schema and

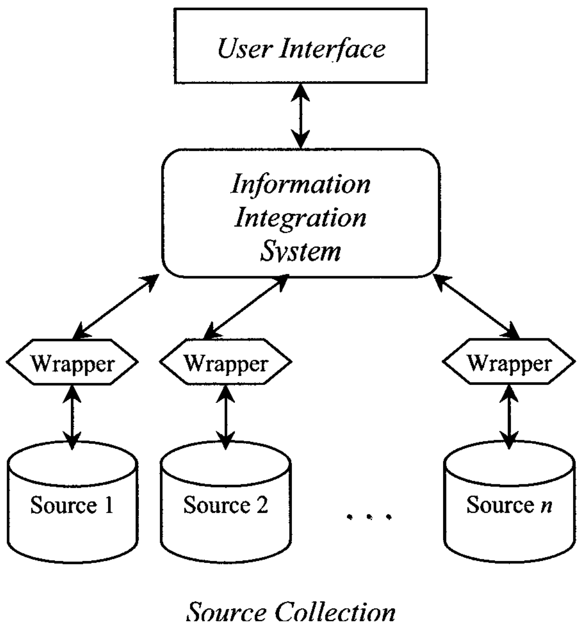

Figure 1: General architecture of an information integration system.

query interfaces information integration systems use so called *mediation architecture.* In this architecture a wrapper is build on the top of each source. The purpose of the wrapper is to translate the source data model to a universal data model, typically *relational model*, so that the system can exchange queries and data with the sources. The general architecture of an information integration system is illustrated by Figure 1.

To provide a uniform interface, an information integration system present to the user a *global schema.* A global schema is a set of *virtual relations*, virtual in the sense that they are not actually stored anywhere. Internally an information integration system keeps information about a set of available *data sources*, that is a set of relations which contain the real data. To be able to answer queries the system must also maintain a set of *mappings* between the global schema and data sources. Users of an information integration system pose queries in terms of the global schema and, therefore, the system should be able to re-express the queries in terms of a suitable set of queries posed to the sources. The crucial step in this reformulation process is deciding how to decompose and assemble a query on the global schema into a set of subqueries on the sources, based on the meaning of the mapping between them. The computed subqueries are then shipped to the sources, and the results are assembled into the final answer.

Critical issue in the design of an information integration system is how the global schema and the source schema are mapped to each other. In particular, two basic approaches have been proposed: the *Global-As-View*(GAV) and the *Local-As-View* (LAV).

The first one, also known as *global-centric approach*, requires that the global schema is expressed in terms of the data sources. More precisely, to every element of the global schema, a view over the data sources is associated, so that its meaning is specified in terms of the data residing at the sources. The second approach, also known as *source-centric approach*, requires the global schema to be specified independently form the sources. In turn, the sources are defined as views over the global schema. The relationships between the global schema and the sources are, thus, established by specifying the information content of each source in terms of a view over the global schema. Comparisons of the two approaches are reported in [Ull97, AGL01].

It is well known that query processing is much easier in the GAV approach, where we can take advantage of the fact that the mapping directly specifies which source queries corresponds to the elements of the global schema and, thus, answering a query basically means unfolding its atoms according to their definition in terms of the sources. The reason why unfolding does the job is that the mapping essentially specifies a single database satisfying the global schema, and evaluating the query over this unique database is equivalent to evaluating its unfolding over the sources. However, in the GAV approach the system lacks an essential feature – extensibility. If a new source is to be added to a GAV information integration system the global schema needs to be redesigned, which is done by and large manually and, therefore, is very costly.

It is also an accepted fact that processing queries in the LAV approach is a difficult task [ALM02, DG97, GM99, LRO96, MM01, PL00]. Indeed, in this approach, the only knowledge we have about the data in the global schema is through the views representing the sources, and these views provide only partial information about the data. Since the mapping associates to each source a view over the global schema, it is not immediate to infer how to use the sources in order to answer queries expressed over the global schema. Thus, extracting information from the LAV information integration system is similar to query answering with incomplete information, which is a complex task [Gra91, IL84, Men84]. On the other hand, the LAV approach ensures an easier extensibility of the integration system, and provides a more appropriate setting for its maintenance. For example, adding a new source to the system requires only providing the definition of the source and does not generally involve changes to the global schema.

In this work we focus on LAV information integration system, where query processing can, as we already mentioned, be seen as a special case of query answering with incomplete information. To see that this is indeed the case consider the following simple example.

Example 1 Suppose we have a global relations *Teaches(Prof, Course)*, and two sources: Source S_1 has definition $S_1(X,Y) \leftarrow$ *Teaches(X,Y)* and contains one fact

S_1(*Johns, comp248*)}. Source S_2 has definition $S_2(X) \leftarrow$ *Teaches(X,Y)* and also one fact S_2(*Smith*)}. Then it is natural to think that any database that contains at least the facts *Teaches(Johns, comp248)*, and *Teaches(Smith, c)*, for some course number c, is a possible global database. For example {*Teaches(Johns, comp248)*, *Teaches(Smith, comp335)*} and {*Teaches(Johns, comp248)*, *Teaches(Smith, comp335)*, *Teaches(Brown, comp353)*} are both examples of such databases.

That is, there might be *several* (usually infinitely many) global databases that are consistent with the definition of, and the data in the sources. In other words, the global database is an incomplete database.

A question of semantics now arises: what is the meaning of a query? As illustrated by the Figure 2 the user that is presented with the global schema expresses his/her query in terms of this schema. Since the sources implicitly represent an instance of this global schema, it would be natural –at least conceptually– to reconstruct the global database represented by the sources and apply the query to this global database.

However, there are at least two issues that must be resolved for this to work. First, what database or databases are represented by a given set of sources, how to identify them, and how to concisely represent this possibly infinite set. Second, since the global database is a set, each element representing a possible global database, the natural answer to a query is the *set* of answers obtained by applying the query on each possible database. Thus, we need to a way to compute a single compact representation of these multiple answers or an approximation to them. See [GM99, MM01, GK02] for the comprehensive treatment of these issues.

From the algorithmical standpoint query answering in LAV information integration system reduces to view-based query processing.

There are two approaches to view-based query processing, called *query rewriting*

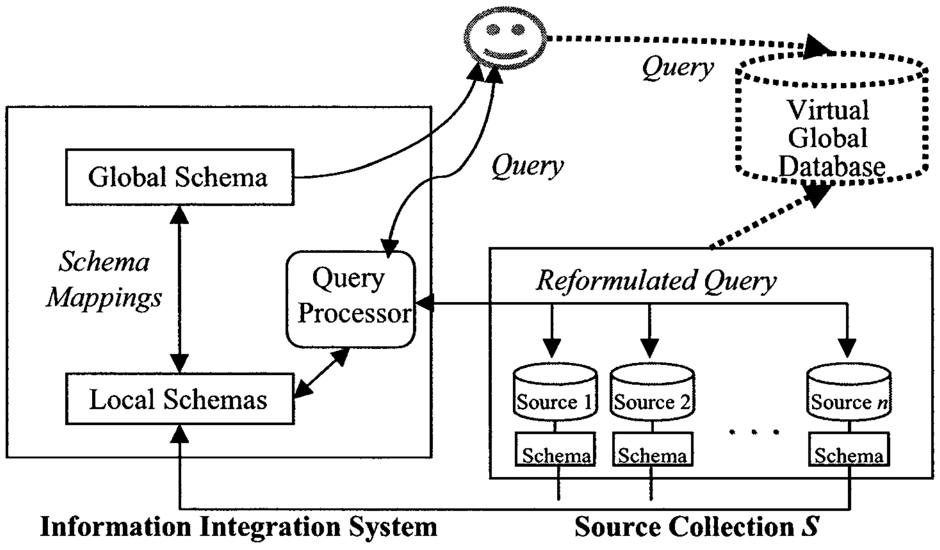

Figure 2: Querying an information integration system.

and *query answering* respectively [CGLV00]. In the former approach, we are given a query and a set of view definitions, and the goal is to reformulate the query into an expression, the *rewriting*, that refers only to the views, and provides the answer to the query. In the latter approach, besides the query and the view definitions, we are also given the data stored at the sources. The goal is to compute the answer that is implied by the sources, i.e., the set of tuples that are in the answer set of the query in all the databases that are consistent with the sources.

Notice the difference between the two approaches. In query rewriting, query processing is divided into two steps, first the query is re-expressed in terms of the view definitions, and then the obtained rewriting is evaluated on the underlying sources. In query answering, we essentially compute the representation of the entire global database and then evaluate the query on this representation. While conceptually clear and, thus, useful for defining semantics of a query in an information integration system, query answering is not practical. Computing the representation of the entire global database might involve a large amount of redundant work, since the global relations that are in the query might be mentioned in only few source definitions. Moreover, the query might have selections and joins that could be computed directly at the sources. For these reasons, mainstream of research in information integration focuses on query rewriting rather then just query answering.

1.2 Problem definition and our contributions

State of the art query processing algorithms in LAV information integration systems all produce what has been thought of as "the best obtainable" answer, given the circumstances that the source-centric approach introduces incomplete information into the virtual global relations. However, perhaps since the relationship between information integration and incomplete information had not been clearly articulated, this "best obtainable" answer does not allow partial information. To illustrate the problem and to make our discussion more concrete let us consider a simple example.

Example 2 Suppose the global schema contains two relations:

> *Prof(Prof, Email, Office, Area)*, that is professor's name, email, office number, and research area.
>
> *Dept(Prof, D)*, that is professor's name and department.

The available sources are $\{S_1, S_2, S_3, S_4\}$. These sources have the following definitions:

> $S_1(Prof, Email) \leftarrow Prof(Prof, Email, Office, Area)$
>
> $S_2(Prof, Office) \leftarrow Prof(Prof, Email, Office, Area)$
>
> $S_3(Prof, Area) \leftarrow Prof(Prof, Email, Office, Area)$

$$S_4(\mathit{Prof}, D) \leftarrow \mathit{Dept}(\mathit{Prof}, D)$$

Suppose the user issues the query

$$Q(\mathit{Prof}, \mathit{Email}, \mathit{Office}, \mathit{Area}) \leftarrow \mathit{Prof}(\mathit{Prof}, \mathit{Email}, \mathit{Office}, \mathit{Area}), \mathit{Dept}(\mathit{Prof}, \mathit{compsci})$$

That is, the user is interested in obtaining all available information about professors in the *compsci* department. Since the information integration system does not have a way to get tuples for the atom *Prof* it would produce an empty rewriting and, thus, an empty answer for the user.

Everybody who has ever queried integrated information, especially in the case of the World Wide Web, knows this situation very well. The next logical action of the user, who wants to get at least partial information, is to try to make his(her) query less restrictive. In our example, to get at least some information about the professors in the *compsci* department, the user has to modify the original query by projecting out some of the attributes of the relation *Prof.* Since the user in unaware of the internals or the system, (s)he has to try all possible combinations of the attributes and then manually assemble the final answer. From the point of view of the user, the system should take on this burden and compute the answer containing this partial information. This is feasible, since the query could be rewritten as union of the following unsafe conjunctive queries.

$$Q(\mathit{Prof}, \mathit{Email}, X, Y) \leftarrow S_1(\mathit{Prof}, \mathit{Email}), S_4(\mathit{Prof}, \mathit{compsci})$$

$$Q(\mathit{Prof}, X, \mathit{Office}, Y) \leftarrow S_2(\mathit{Prof}, \mathit{Office}), S_4(\mathit{Prof}, \mathit{compsci})$$

$$Q(\mathit{Prof}, X, Y, \mathit{Area}) \leftarrow S_3(\mathit{Prof}, \mathit{Area}), S_4(\mathit{Prof}, \mathit{compsci})$$

The unrestricted variables X and Y represent unknown values. The answer can then be presented to the user as a table with some values missing, for example as

Pname	Email	Office	Area
Murphy	murphy@cs.uoft.edu	$\perp$	$\perp$
Murphy	$\perp$	SF322	$\perp$
Smith	$\perp$	$\perp$	DB
Jones	jones@cs.concordia.ca	$\perp$	$\perp$
Brown	$\perp$	$\perp$	AI

Figure 3: One possible answer to the query in the Example 2.

the table in Figure 3. (The contents of the table will, obviously, depend on the data provided by the sources.)

However, producing such partial answers is not without its intricacies. In a web based setting, the number of tuples in the answer is typically is much larger than the user wants. In such a case the user, to reduce the size of the answer, would most likely want to make his(her) query more restrictive. Which means that an information integration system, in order to be of any practical use, has to provide the user with the ability to search within the answers. This poses the restriction on how the answer to the original query is to be computed. Semantics of the query should be compositional, namely, if the second query is applied to the results of the first one the answer should be the same as if the composition of the two queries were executed on the underlying data sources.

In this work we solve both problems illustrated by the above example.

We first define the semantics of partial facts and introduce the notion of an exact answer - that is an answer that includes partial facts. We then provide two methods for computing the exact answer, in such way that semantics of queries remain compositional.

The first method is a generalization of the "inverse-rules" approach [DG97, GM99, MM01], and involves explicitly computing a syntactic representation of the set of global databases implicitly defined by the sources. The other method is a generalization of the rewriting technique (see e.g. [LMSS95, Ull97, LRO96, PL00]). In this method reformulates a query in terms of the source relations, and, thus, avoids inverting the entire source collection.

Current research in information integration focuses on computation of *possible* or, most commonly, the *certain* answer. Possible answer is an approximation of the exact answer from above and certain answer is an approximation of the exact answer from below. The ability to compute exact answer allows us to construct either approximation easily since the certain answer is the intersection of exact answers, and the possible answer corresponds to their union.

The outline of the thesis is as follows. In Chapter 2 we begin by introducing some notions, defining the concepts, and formalizing the model that are used in the thesis. Chapter 3 deals with the issue of query answering and query rewriting. We first address the question of semantics of the query in a LAV information integration system. Then, Section 3.1 gives the query answering algorithm and Section 3.2 presents two query rewriting algorithms. Chapter 4 gives the results of experimental evaluation of the performance of the two proposed rewriting algorithms. Chapter 5 reviews main contributions of our work, outlines directions for future research, and concludes.

Some of the material presented in this thesis appears in [GK02], [GK03].

Chapter 2

The Model

2.1 Basic definitions

We begin by introducing some notions and defining the concepts that are used throughout the thesis.

Let **rel** be a countably infinite set $\{R, S, \ldots, R_1, S_1, \ldots\}$ of *relation names*, let **dom** be a countably infinite set of *constants*, and let **var** be a countably infinite set of *variables*. Constants will be denoted by lower case letters and variables by upper case letters. Associated with each relation name R is a positive integer $arity(R)$, which is the arity of R.

A *fact* over R is an expression of the form $R(a_1, \ldots, a_k)$, where $k = arity(R)$, and each a_i is in **dom**. Let $\mathbf{R} = \{R_1, R_2, \ldots, R_n\}$ be a set of relation names. A finite set of relation names will sometimes also be called a *schema*. A *database db* over $\mathbf{R}$ is a finite set of facts, each fact being over some $R_i \in \mathbf{R}$.

As the majority of papers on information integration do, we will focus on the most practical class of queries, namely the conjunctive queries. To formally define a conjunctive query we need the concept of an atom, which is generalization of a fact in that it may include variables as well as constants. An *atom* over a relation name R is an expression of the form $R(e_1, \ldots, e_k)$, where $k = \mathit{arity}(R)$, and each e_i is either a constant in **dom** or a variable in **var**.

A *conjunctive query* φ (over $\mathbf{R}$) has the form

$$\mathit{head}(\varphi) \leftarrow \mathit{body}(\varphi),$$

where $\mathit{body}(\varphi)$ is a set of atoms $b_1, b_2, \ldots, b_n$, each over a relation name in $\mathbf{R}$, and $\mathit{head}(\varphi)$ is an atom over an answer relation name not in $\mathbf{R}$. We assume that all variables occurring in $\mathit{head}(\varphi)$ also occur in $\mathit{body}(\varphi)$, i. e. that the query φ is *safe*. The variables occurring in $\mathit{head}(\varphi)$ are the *distinguished* variables of the query, and all the others are *existential* variables.

A conjunctive query φ can be applied to a database db over $\mathbf{R}$, resulting in a set of facts

$$\varphi(db) = \{\sigma(\mathit{head}(\varphi)) : \sigma(\mathit{body}(\varphi)) \subseteq db \text{ for some valuation } \sigma\}.$$

A *valuation* σ, is formally a finite partial mapping from $\mathbf{var} \cup \mathbf{dom}$ to **dom** that is the identity on **dom**. Valuations, like $X_i \mapsto a_i$, for $i \in [1, p]$, we will give in the form $\{X_1/a_1, \ldots, X_p/a_p\}$. The identity on constants is omitted in this notation.

2.2 Source collections and global databases

Let **loc** be a countably infinite set $\{V, V_1, V_2, \ldots\}$ of *local relation names.* The local relation names have arities, and atoms over local relation names are defined in the same way as atoms over relation names in **rel**. To distinguish between relations (relation names) in **rel** and in **loc** we will henceforth call the former *global* relations (relation names).

A *source* S is a pair (φ, v), where φ is a conjunctive query and v is a finite set of facts over $head(\varphi)$. A *source collection* $\mathcal{S}$ is a finite set of sources. The *(global) schema* of $\mathcal{S}$, denoted $sch(\mathcal{S})$ is the set consisting of all the global relation names occurring in the bodies of the defining conjunctive queries of the sources in $\mathcal{S}$. The *description* of $\mathcal{S}$, denoted $desc(\mathcal{S})$ is obtained from $\mathcal{S}$ by dropping the extension from every pair in $\mathcal{S}$. In other words, a source collection $\mathcal{S}$ has *two* "schemas," $sch(\mathcal{S})$ which is the "global world view" and $desc(\mathcal{S})$, which describes the defining views. The *extension* of a source collection $\mathcal{S}$, denoted $ext(\mathcal{S})$, is the union of all facts in sources in $\mathcal{S}$.

Since the global relations are virtual, in the sense that they don't contain any data, we need to answer the question: what is the instance of the global schema that is implicitly represented by the sources? As we already discussed in the Chapter 1, the global database is actually incomplete. In other words, there might be several (usually infinitely many) global databases that are consistent with the definition of, and the data in the sources (see Example 1).

Formally, a source collection $\mathcal{S}$ defines a set of possible databases, denoted $poss(\mathcal{S})$, as follows:

$$\begin{aligned} poss(\mathcal{S}) = \{db \ \text{ over } \ sch(\mathcal{S}) \quad : \quad & v_i \subseteq \varphi_i(db) \ \text{ for all sources} \\ & S_i = (\varphi_i, v_i) \text{ in } \mathcal{S}\}. \end{aligned}$$

Since $poss(\mathcal{S})$ is infinite, we will now consider the problem of finitely representing an infinite set of databases. For this we invoke the venerable tableau.

Sets of databases and tableaux

Tableaux [Men84] are intended to concisely and finitely represent a large or infinite set of possible instances.

Let $\mathbf{R} = \{R_1, R_2, \ldots, R_n\}$ be a set of relation names. A *tableau* T over $\mathbf{R}$ is a finite set of atoms over the R_i's. Note that the same variable might appear in several atoms in T.

A tableau T over schema $\mathbf{R}$ represents a set of databases over $\mathbf{R}$. This set is denoted $rep(T)$, and it is defined by

$$rep(T) = \{db \ : \text{ there is a valuation } \sigma \text{ such that } \sigma(T) \subseteq db\}.$$

The definition says that a database db is represented by a tableau T, if there is a valuation σ such that when all variables in T are replaced by their image under σ, the set of facts thus obtained is a subset of db (see Example 3).

Example 3 Let $T = \{R(a,b), R(c,X)\}$, $db = \{R(a,b), R(c,d), R(e,f)\}$, and $db' = \{R(a,g), R(c,d)\}$. Then db is in $rep(T)$, but db' is not. $R(a,b)$ in db is a certain fact because it's true in all databases in $rep(T)$ and $R(c,d)$ is so called possible fact because it's true in some databases in $rep(T)$.

Note that $\sigma(T)$ is a *subset* of db. this is due to the fact that we adopt the *Open World Assumption* (OWA) [Rei78]. Usually a database is a complete description of the state of the world modeled by it. This is actually true only if we adopt the *Closed World Assumption* (CWA) according to which facts not explicitly stored in the database are false. For example, in the database $\{R_1(a_1,a_3), R_1(a_2,a_3), R_2(a_3,a_4)\}$, the fact $R_1(a_3,a_1)$ is false. The closed world assumption is convenient since in general there is an infinitude of false facts. The CWA is, however, not appropriate for modeling all situations and information integration is a setting that calls for use of the OWA. The OWA regards the database as an *incomplete* description of the world: All facts stored in the database are true, the truth value of any other fact is *unknown.* Semantically this means that the stored database db actually is a finite description of a *set of possible worlds*, defined as

$$\{db' : db \subseteq db'\}. \tag{1}$$

Each db' represents one possible complete state of affairs (or world). Facts that are true in *some* possible worlds are called *possible* facts, and facts true in *all* possible worlds are called *certain* facts.

In order to reason about tableaux we have to define a few concepts that are applicable to sets of global databases represented by tableaux.

Let $\mathcal{X}$ and $\mathcal{Y}$ be two enumerable sets of global databases over $\mathbf{R}$. We say that $\mathcal{X}$ and $\mathcal{Y}$ are *coinitial* if they have the same $\subseteq$-minimal elements. Coinitiality is denoted $\mathcal{X} \approx \mathcal{Y}$.

Let Ω be the set of all queries expressible in a query language that we by abuse of notation also call Ω. Then $\mathcal{X}$ and $\mathcal{Y}$ are said to be *Ω-equivalent*, denoted $\mathcal{X} \equiv_\Omega \mathcal{Y}$, if for all queries $\varphi \in \Omega$ we have

$$\bigcap_{db \in \mathcal{X}} \varphi(db) = \bigcap_{db \in \mathcal{Y}} \varphi(db).$$

The intuition behind Ω-equivalence is that $\mathcal{X}$ and $\mathcal{Y}$ are indistinguishable as far as the certain information derivable by queries in Ω are concerned. Thus, if a user can only pose queries in Ω, (s)he cannot distinguish between $\mathcal{X}$ and $\mathcal{Y}$.

Let db and db' be two databases over $\mathbf{R}$. A query language Ω is *monotone* if for every query $\varphi \in \Omega$ $\varphi(db) \subseteq \varphi(db')$, whenever $db \subseteq db'$.

The following lemma is proved in the seminal paper [IL84].

Lemma 1 *Let Ω be a monotone query language. If $\mathcal{X} \approx \mathcal{Y}$, then $\mathcal{X} \equiv_\Omega \mathcal{Y}$.*

Proof. Let $\mathcal{X} \approx \mathcal{Y}$, and let db be an arbitrary element in $\mathcal{X}$. We have two cases to consider.

Case 1: The element db is $\subseteq$-minimal in $\mathcal{X}$. Then there exist $db' \in \mathcal{Y}$ such that $db \subseteq db'$. Now, by the definition of monotonicity, for every query $\varphi \in \Omega$, we have $\varphi(db) \subseteq \varphi(db')$. Consequently $\cap\{\varphi(db) : db \in \mathcal{X}\} \subseteq \cap\{\varphi(db) : db \in \mathcal{Y}\}$

Case 2: The element db is not $\subseteq$-minimal in $\mathcal{X}$. Then there is a $\subset$-minimal element $db' \in \mathcal{X}$ such that $db' \subseteq db$, and $db' \subseteq db''$, for some $db'' \in \mathcal{Y}$. Now we have that $\varphi(db) \cap \varphi(db') \subseteq \varphi(db'')$. Hence again we get, $\cap\{\varphi(db) : db \in \mathcal{X}\} \subseteq \cap\{\varphi(db) : db \in \mathcal{Y}\}$.

The same reasoning applies for showing that $\cap\{\varphi(db) : db \in \mathcal{Y}\} \subseteq \cap\{\varphi(db) : db \in \mathcal{X}\}$. ■

Of particular interest to us is of course choosing Ω to be the set of all unions of conjunctive, which, it goes without saying, is a monotone query language.

2.3 Representing $poss(\mathcal{S})$ by a tableau

Now the set $poss(\mathcal{S})$ can be conveniently represented by a tableau over schema $sch(\mathcal{S})$, denoted $T(\mathcal{S})$, such that $rep(T) = poss(\mathcal{S})$. To construct T we shall follow the approach in [GM99]. We define a function, which, by abuse of notation, we by also denote T, from sources with defining view φ, where the body of φ consists of atoms over relation names in $\mathbf{R}$, to a tableau over $\mathbf{R}$. We also need an auxiliary function *refresh*, that, when applied to a set of atoms, replaces all variables with distinct ones. Given a source $S = (\varphi, v)$, we set

$$T(S) = \bigcup_{u \in v} \{refresh(\sigma(body(\varphi))) \quad : \quad \sigma(head(\varphi)) = u\},$$

for some valuation σ.

Example 4 Let $S = (V(X,Z) \leftarrow R(X,Y), S(Y,Z), \{V(a,b), V(c,d)\})$. Then $T(S) = \{R(a,Y_1), S(Y_1,b),\ R(c,Y_2), S(Y_2,d)\}$, where Y_1 and Y_2 are fresh variables.

When there are several sources in $\mathcal{S}$ we set

$$T(\mathcal{S}) = \bigcup_{S \in \mathcal{S}} T(S).$$

The tableau constructed by the function T has the following desirable property.

Theorem 1 *$rep(T(\mathcal{S})) = poss(\mathcal{S})$.*

Proof.

Let $db \in rep(T(\mathcal{S}))$. To prove that $db \in poss(\mathcal{S})$ we need to show that for all sources $S_i = (\varphi_i, v_i)$ in $\mathcal{S}$, we have $v_i \subseteq \varphi_i(db)$. Since $db \in rep(T(\mathcal{S}))$ there is a valuation σ such that $\sigma(T(\mathcal{S})) \subseteq db$. Let $S_i = (\varphi_i, v_i)$ be an arbitrary source in $\mathcal{S}$, and let t be an arbitrary fact in v_i. Then there must be a substitution θ, such that $t = \theta(head(\varphi_i))$ and all facts in $\theta(body(\varphi_i))$ are in $T(\mathcal{S})$. It follows that $\theta(\sigma(body(\varphi_i))) \subseteq db$ and, consequently, $\theta(\sigma(head(\varphi_i))) \in db$. Since $\theta(\sigma(head(\varphi_i))) = t$, we have $v_i \subseteq \varphi_i(db)$ as desired.

For inclusion in the other direction, let $db \in poss(\mathcal{S})$. From construction of $T(\mathcal{S})$ it immediately follows that there is a valuation σ such that $\sigma(T(\mathcal{S})) \subseteq db$ and, thus, that $db \in rep(T(\mathcal{S}))$. ■

Note that $T(\mathcal{S})$ can be extended to handle the closed-world assumption and other kinds of constraints, see [GM99, MM01].

Now, since we laid out a formal framework for information integration and expounded the strong connection between information integration and incomplete information we can address the question of querying in an information integration system.

Chapter 3

Query Answering and Query Rewriting

3.1 Query answering

First we have to address the question of semantics of the query in information integration system. Since the global database is incomplete, and therefore, represents a set of possible databases, it would be natural to expect that the answer to a query is also a set of answers, one for each possible database. With this in mind we define exact answer.

Let $\mathcal{S}$ be source collection, and φ a conjunctive query, such that the body of φ consists of atoms over relation names in $sch(\mathcal{S})$. Now φ applied to $\mathcal{S}$ defines the *exact answer*:

$$\varphi(\mathcal{S}) = \{\varphi(db) \ : \ db \in poss(\mathcal{S})\}.$$

The definition essentially says that since the source collection corresponds to a set of databases, the answer should also correspond to a set, obtained by evaluating the query pointwise.

The problem of computing exact answer to a user query was not addressed in the literature except for a brief discussion in [GM99] and somewhat more extensive treatment in [GK02]. Instead, all of the proposed algorithms are aimed at computing the possible or, most commonly, the certain answer, where the possible answer is an approximation of the exact answer from above and the certain answer is an approximation of the exact answer from bellow.

The *possible answer* can be defined as $\varphi^*(\mathcal{S}) = \bigcup\{\varphi(db) : db \in poss(\mathcal{S})\}$.

The *certain answer* can be defined as $\varphi_*(\mathcal{S}) = \bigcap\{\varphi(db) : db \in poss(\mathcal{S})\}$.

Essentially, the exact answer is the most general answer, and, as we show in the following section, given the exact answer we can obtain both possible and certain answers.

Next we will discuss how to compute the exact answer and to produce a single compact representation of it, so that semantics of the queries remain compositional, that is, if we desire to apply a query to the result of an earlier query we obtain an answer that we expect.

3.1.1 Computing the exact answer from the tableau

Since we are able to construct a tableau T representing all databases in $poss(\mathcal{S})$ it is natural to extend the standard query evaluation mechanism to operate on tableaux.

Given a conjunctive query φ over $\mathbf{R}$, our evaluation $\widehat{\varphi}$ (which is basically the "naive evaluation" of [IL84]) is defined next. For this we need the concept of a substitution. A *substitution* is a valuation, except that we allow variables to be mapped into variables, not only constants. Thus, a substitution θ is a function from (a subset of) $\mathbf{dom} \cup \mathbf{var}$ to $\mathbf{dom} \cup \mathbf{var}$, keeping in mind that constants have to be mapped to themselves. Then

$$\widehat{\varphi}(T) = \{\theta(\mathit{head}(\varphi)) : \theta(\mathit{body}(\varphi)) \subseteq T \text{ for some substitution } \theta\}.$$

Example 5 Let a query be $\varphi = Q(X,Y,Z) \leftarrow R(X,Y), S(Y,Z)$ and a tableau be $T = \{R(a,b), R(d,X), S(b,c), S(X,e), S(Y,f)\}$. Then $\widehat{\varphi}(T) = \{Q(a,b,c), Q(d,X,e)\}$.

Clearly, if our definition of $\widehat{\varphi}$ is semantically meaningful, then we should expect that it approximates the information given by $\varphi(\mathcal{S})$ in some natural sense. Indeed, our extended semantics has the following property:

Theorem 2 *$rep(\widehat{\varphi}(T)) \approx \varphi(rep(T))$.*

Proof.

Let db be a $\subseteq$-minimal element in $rep(\widehat{\varphi}(T))$. Then there exist a valuation σ such that $\sigma(\widehat{\varphi}(T)) = db$. Let t be an arbitrary fact in db. Then there is a fact $u \in \widehat{\varphi}(T)$ such that $\sigma(u) = t$, and there is a substitution θ such that $\theta(\mathit{body}(\varphi)) \in T$ and $u = \theta(\mathit{head}(\varphi))$.

Let σ' be an extension of σ that maps every variable that is in T but not in $\widehat{\varphi}(T)$ to a distinct new constant. Since $\theta(\mathit{body}(\varphi)) \subseteq T$, we have $\sigma'\theta(\mathit{body}(\varphi)) \subseteq \sigma'(T)$. It now follows that $t = \sigma'(\theta(\mathit{head}(\varphi))) \in \varphi(\sigma'(T))$. Note that $\sigma'(T)$ is a $\subseteq$-minimal element in $rep(T)$. From the monotonicity of φ it follows that $\varphi(\sigma'(T))$ is a $\subseteq$-minimal element in $\varphi(rep(T))$. We have established that $db \subseteq \varphi(\sigma'(T))$

That concludes the proof that any $\subseteq$-minimal element db in $rep(\widehat{\varphi}(T))$ is also in $\varphi(rep(T))$.

For inclusion in the other direction let db be a $\subseteq$-minimal element in $\varphi(rep(T))$. Then there is a valuation σ, such that $db = \varphi(\sigma(T))$. Let t be an arbitrary tuple in db. Then there is a valuation ρ, such that $t = \rho(\mathit{head}(\varphi))$ and all facts in $\rho(\mathit{body}(\varphi))$ are in $\sigma(T)$. Now we have two cases to consider.

Case 1: The valuation σ is one-to-one. Then there is an inverse σ^{-1}, and hence $\sigma^{-1}(\rho(body(\varphi))) \subseteq \sigma^{-1}(\sigma(T)) = T$, and consequently $\sigma^{-1}(t) = \sigma^{-1}(\rho(head(\varphi)))$ is in $\widehat{\varphi}(T)$. Since $\sigma(\widehat{\varphi}(T)) \in rep(\widehat{\varphi}(T))$, it follows that $t \in \sigma(\widehat{\varphi}(T)) \in rep(\widehat{\varphi}(T))$. Likewise, if t' is any other tuple in $db = \varphi(\sigma(T))$, it is generated by some valuation ρ', and we have $\sigma^{-1}(t') = \sigma^{-1}(\rho'(head(\varphi))) \in \widehat{\varphi}(T)$. Therefore $db \subseteq \sigma(\widehat{\varphi}(T))$.

Case 2: There is (at least one) pair of distinct variables X and Y in T, such that $\sigma(X) = \sigma(Y)$. If $\sigma(X) = \rho(U)$, and $\sigma(Y) = \rho(W)$, for $U \neq W$, then the valuation ω, that is like $\sigma^{-1} \circ \rho$, except $\omega(U) = X$, and $\omega(V) = Y$, gives us $\omega(body(\varphi)) \subseteq T$, and $\sigma^{-1}(t) = \omega(head(\varphi)) \in \widehat{\varphi}(T)$. Consequently $t = \sigma(\sigma^{-1}(t)) \in \sigma(\widehat{\varphi}(T))$.

Suppose then that $\sigma(X) = \sigma(Y) = \rho(W)$, and that there are (at least) two occurrences of W in $body(\varphi)$. Consider now the valuation σ', that is exactly like σ, except it maps Y to a fresh constant, say a. Clearly $t \notin \varphi(\sigma'(T))$, and any fact in $\varphi(\sigma'(T))$ is also in $\varphi(\sigma(T))$ (because there is an embedding of $\sigma'(T)$ into $\sigma(T)$.) Therefore we have a contradiction to the assumption that t belonged to a $\subseteq$-minimal element of $\varphi(rep(T))$. ■

As a consequence we now have a method for computing an $\approx$-approximation of $\varphi(\mathcal{S})$.

Corollary 1 *$rep(\widehat{\varphi}(T(\mathcal{S}))) \approx \varphi(\mathcal{S})$.* ■

In other words, first invert the source extensions through their definitions, then apply the $\widehat{\varphi}$-evaluation of the user query φ on the resulting tableau. The result of the evaluation is another tableau, which the user perceives as a relation with nulls. Given the exact answer that is obviously most informative of all answers, we can obtain the possible answer and the certain answer as follows.

Lemma 2 *$\varphi_*(\mathcal{S}) = \cap\, rep(\widehat{\varphi}(T(\mathcal{S})))$, and $\varphi^*(\mathcal{S}) \approx \cup\, rep(\widehat{\varphi}(T(\mathcal{S})))$.*

Proof.

To prove the first clam of the lemma, recall that by the definitions of $\varphi(\mathcal{S})$ and $\varphi_*(\mathcal{S})$, $\varphi_*(\mathcal{S}) = \cap\varphi(\mathcal{S})$. By the Theorem 2 $rep(\widehat{\varphi}(T)) \approx \varphi(rep(T))$. Now, since φ is monotone, by the definition of Ω-equivalence and Lemma 1 $\varphi_*(\mathcal{S}) = \cap\, rep(\widehat{\varphi}(T(\mathcal{S})))$.

Analogous proof for the second claim is omitted. ■

Note that the above lemma is not surprising, since query plan produced by "inverse rules" algorithm of [DG97], at least conceptually, computes $T(\mathcal{S})$ by inverting entire source collection and, then, considers only the certain tuples for the answer.

Furthermore, we can prove that $\widehat{\varphi}$ evaluation has compositional semantics.

Lemma 3 $rep(\widehat{\varphi}(\widehat{\psi}(T(\mathcal{S})))) \approx \varphi(\psi(\mathcal{S}))$.

Proof.

From Theorem 2 follows that $rep(\widehat{\varphi}(\widehat{\psi}(T(\mathcal{S})))) \approx \varphi(rep(\widehat{\psi}(T(\mathcal{S}))))$. By Corollary 1 $rep(\widehat{\psi}(T(\mathcal{S}))) \approx \psi(\mathcal{S})$. Therefore, $rep(\widehat{\varphi}(\widehat{\psi}(T(\mathcal{S})))) \approx \varphi(\psi(\mathcal{S}))$. ■

In essence, as illustrated by Example 6, we can use our extended evaluation to compute a query on the result of another query.

Example 6 For a simple example, let us consider the following source collection $\mathcal{S} = \{(V_1(X_1, X_3, X_4, X_5) \leftarrow R(X_1, X_2, X_3), R(X_4, X_2, X_5), \{V_1(a, b, c, d)\})\}$, the query $Q_1(X, Y, Z) \leftarrow R(X, Y, Z)$, and the query $Q_2(X, W) \leftarrow R(X, Y, Z)R(W, Y, U)$. The exact answer to Q_1 is $\{Q_1(a, X, b), Q_1(c, X, d)\}$. The result of applying Q_2 on the exact answer to Q_1 is $\{Q_2(a, c)\}$ and it is easy to see that it is the same as if Q_2 was evaluated directly.

However, computing $\widehat{\varphi}(T(\mathcal{S}))$ might involve abundance of redundant work, since it amounts to constructing the tableau corresponding to the entire source collection, whereas the global relations that are in $body(\varphi)$ might be mentioned in only few source definitions. Moreover, the query might have selections and joins that could be computed directly at the sources. For those reasons, vast majority of research in information integration focuses on query rewriting rather then just query answering. We discuss query answering by the means of query rewriting in the next section.

3.1.2 Computing the exact answer directly on the source collection

In an information integration system a query processor is in charge of reformulating queries written in terms of the global schema to queries on the appropriate sources. This process is known as *query rewriting.* To obtain the answer to a given query the rewriting is then evaluated on the underlying sources. Since the exact answer is actually a set of answer sets and can be represented by a tableau, any rewriting to compute the exact answer has to be more general than the usual notion of rewriting.

We now proceed as follows. First we generalize the notion of query containment, which enables comparisons between different reformulations of queries, to *p-containment.* Next we define *p-rewriting* based on this more general containment. Then we extend standard query evaluation so that given a p-rewriting it computes the exact answer. In Section 3.2 we study the problem of finding p-rewriting and its complexity and give two algorithms that compute a p-rewriting for a given query.

P-containment of conjunctive queries

We can broaden query containment as follows.

Let φ_1 and φ_2 be conjunctive queries. A query φ_1 is said to be *p-contained* in φ_2, denoted $\varphi_1 \subseteq_p \varphi_2$, if and only if there exists a conjunctive query ϕ, where φ_1 is equivalent to $\pi_L(\phi)$ (π is relational projection), for some list L of columns in $head(\phi)$ taken in the original order, such that for all databases db, $\phi(db) \subseteq \varphi_2(db)$. Note that p-containment is a generalization of query containment since L can be the list of all columns in φ_2.

Testing p-containment of conjunctive queries

In order to facilitate testing of p-containment, the classical notion of a containment mapping can be generalized to define p-containment mappings. A *p-containment mapping* from a conjunctive query φ_2 to a conjunctive query φ_1 is a mapping μ, from variables of φ_2 to variables and constants of φ_1, such that

1. $\mu(body(\varphi_2)) \subseteq body(\varphi_1)$, and
2. for every variable X in $head(\varphi_1)$ there is a variable Y in $head(\varphi_2)$, such that $\mu(Y) = X$.

Example 7 Consider the following queries

$$\varphi_1 = Q_1(X) \leftarrow R(X,Y), S(Y,Y), T(Y,Z)$$

$$\varphi_2 = Q_2(A,B) \leftarrow R(A,B), S(B,C)$$

There is a p-containment mapping $\mu = \{A/X, B/Y, C/Y\}$ from φ_2 to φ_1.

As consequence, we can now use p-containment mappings to test p-containment of conjunctive queries.

Theorem 3 *A query φ_1 is p-contained in a query φ_2 if and only if there is a p-containment mapping from φ_2 to φ_1.*

Proof.

Let μ be a p-containment mapping from φ_2 to φ_1, and let db be an arbitrary database. A fact t_1 in $\varphi_1(db)$ is generated by some valuation σ. Then $\sigma \circ \mu$ is a valuation that generates the corresponding fact t_2 in $\varphi_2(db)$. To see that this is indeed so, let $b \in body(\varphi_2)$. Then $\sigma \circ \mu(b) = \sigma(c) \in db$, for some $c \in body(\varphi_1)$. Therefore, $\sigma \circ \mu(b) \in db$. From requirement 2 of a p-containment mapping it follows that $\sigma \circ \mu(head(\varphi_2)) = \pi_L(\sigma \circ \mu(head(\varphi_1)))$, where L is a list of variables in $head(\varphi_2)$ in the original order. Thus, $\varphi_1 \subseteq_p \varphi_2$.

Let $\varphi_1 \subseteq_p \varphi_2$. Let db be the canonical database that is the "frozen" $body(\varphi_1)$. By the definition of p-containment there exist a conjunctive query ψ, such that $\psi(db) \subseteq \varphi_2(db)$ and $\varphi_1 = \pi_L(\psi)$ for some ordered list L of columns in $head(\psi)$. Obviously, $\varphi_1(db)$ contains a fact t_1, which is the "frozen" $head(\varphi_1)$. Since $\varphi_1 = \pi_L(\psi)$ there must be a fact t_2 in $\psi(db)$, such that $\pi_L(t_2) = t_1$. Since $\psi(db) \subseteq \varphi_2(db)$, we have $t_2 \in \varphi_2(db)$.

Let σ be a valuation that generates the fact t_2 in $\varphi_2(db)$. Let ρ be the the "freezing" mapping, which also is a valuation that generates the fact t_1 in $\varphi_1(db)$. Then $\rho^{-1} \circ \sigma$ is a p-containment mapping from φ_2 to φ_1.

To see that this is indeed so note two things. First, that each atom $b \in body(\varphi_2)$ is mapped by σ to some fact in db, which is a frozen version of some atom $c \in body(\varphi_1)$, so $\rho^{-1} \circ \sigma$ maps b to the unfrozen fact, that is to c itself.

Second, note that those variables in $head(\varphi_2)$ that are also in $head(\varphi_1)$ are mapped by σ to constants in the fact t_1, which is the frozen $head(\varphi_1)$, so that all of the head variables in φ_1 are covered. Thus $\rho^{-1} \circ \sigma$ maps corresponding variables in $head(\varphi_2)$ to the unfrozen $head(\varphi_1)$. Thus, $\rho^{-1} \circ \sigma$ is a p-containment mapping from φ_2 to φ_1.

■

P-rewriting

Now we can extend the notion of rewriting to *p-rewriting*. Let a source collection be $\mathcal{S} = \{S_1(\psi_1, v_1), \ldots, S_n(\psi_n, v_n)\}$ and let φ be a query over $desc(\mathcal{S})$. The *expansion* of φ, denoted φ^{exp}, is defined only if, for each ψ_i in the $body(\varphi)$, there is a containment mapping μ from $head(\psi_i)$ to ψ_i in the $body(\varphi)$. In this case φ^{exp} is obtained from φ by replacing all ψ_i in the $body(\varphi)$ with $\mu(body(\psi_i))$. Existential variables in $\mu(body(\psi_i))$ are replaced by fresh variables in φ^{exp}.

Let $\mathcal{S}$ be a source collection and φ be a conjunctive query over $sch(\mathcal{S})$. The query χ is a *contained rewriting* of φ using $\mathcal{S}$ if $\chi^{exp} \subseteq \varphi$. The query ψ is a *p-contained rewriting* of φ using $\mathcal{S}$ if $\psi^{exp} \subseteq_p \varphi$.

Let ψ be a p-contained rewriting of φ. We define the φ-evaluation of ψ, denoted ψ_φ, as follows.

$$\psi_\varphi(\mathcal{S}) = \{\sigma_\mu(head(\varphi)) : \sigma(body(\psi)) \subseteq ext(\mathcal{S})\},$$

where μ is a p-containment mapping from φ to ψ^{exp} and σ is a valuation. Suppose head variable X of φ is is mapped by μ to a variable in the expansion of atom $V_i(X_1, \ldots, X_k)$, and $\mu(X)$ is the j:th existential variable in the definition of S_i. Then the extension σ_μ of σ, is defined by setting

$$\sigma_\mu(X) = \begin{cases} \sigma(\mu(X)), \text{ if } \mu(X) \text{ occurs in } head(\psi) \\ f_{i,j}(\sigma(X_1), \ldots, \sigma(X_k)), \text{otherwise.} \end{cases}$$

In order to define $\tilde{\varphi}(\mathcal{S})$ we also need an auxiliary function *replace* that when applied to a set of atoms with function terms, replaces each unique term with unique variable. Now we set

$$\tilde{\varphi}(\mathcal{S}) = replace\left(\bigcup\{\psi_\varphi(\mathcal{S}) : \psi^{exp} \subseteq_p \varphi\}\right),$$

and state the following important result:

Theorem 4 *$\widetilde{\varphi}(\mathcal{S}) = \widehat{\varphi}(T(\mathcal{S}))$ up to renaming of the variables.*

Proof.

Let t be an arbitrary atom in $\widetilde{\varphi}(\mathcal{S})$. Then there was conjunctive query ψ (over $desc(\mathcal{S})$) in the union of maximally-contained p-rewritings of φ and a φ-evaluation of ψ, using a valuation σ_μ such that $t = \sigma_\mu(head(\varphi))$, and $\sigma(body(\psi)) \subseteq ext(\mathcal{S})$, where μ is a containment mapping from φ to ψ^{exp}.

Since $\sigma(body(\psi)) \subseteq ext(\mathcal{S})$, it means that all atoms in $\sigma(body(\psi^{exp}))$ are in $T(\mathcal{S})$ (with fresh existential variables). Since μ is a containment mapping from φ to ψ^{exp}, we have that $\sigma(\mu(body(\varphi))) \subseteq T(\mathcal{S})$. Thus $\sigma(\mu(head(\varphi))) \in \widehat{\varphi}(T(\mathcal{S}))$. Now $\sigma(\mu(head(\varphi)))$ is equal to $\sigma_\mu(head(\varphi))$, except for positions that don't occur in $head(\psi)$, these have been replaced by function terms in $\sigma_\mu(head(\varphi))$. Now let us consider the case that a given term appears somewhere else in $\widetilde{\varphi}(\mathcal{S})$. There are two cases, either the repeating term appears in the same atom or it appears in a different atoms. *Case 1:* (*Cf.* Example 8.) The repeating term appears in the same atom. This means that either there were repeating variable in $head(\varphi)$ and it is obvious that in $\widehat{\varphi}(T(\mathcal{S}))$ there is two occurrences of the same variable. Another possibility is that in ψ^{exp} there were two mentions of the same existential variable. In this situation in $T(\mathcal{S})$ there would be two occurrences of the same variables and since $\widehat{\varphi}$ always substitutes variables of the query for the variables of the tableau they would also appear in $\widehat{\varphi}(T(\mathcal{S}))$. *Case 2:* (*Cf.* Example 9.) The repeating term appears in some atom t' in $\widetilde{\varphi}(\mathcal{S})$. In this case there are two possibilities, either t' was produced by φ-evaluation of ψ or by φ-evaluation of some other rewriting ψ'. In either situation both terms originated from the same tuple in $ext(\mathcal{S})$ and, moreover, from the same existential variable because function terms are subscripted with variable index. It is obvious that there would be two occurrences of the same variables in $T(\mathcal{S})$.

For the proof of inclusion in the other direction, let t be an arbitrary atom in $\hat{\varphi}(T(\mathcal{S}))$. Suppose $body(\varphi)$ consists of atoms $b_1, b_2, \ldots, b_n$. Then there is a substitution θ, such that $\theta(b_i) \in T(\mathcal{S})$, for all $i \in \{1, \ldots, n\}$. But each atom $\theta(b_i)$ is in $T(\mathcal{S})$ which follows from the fact that there is a source $S_{i_j} = (\varphi_{i_j}, v_{i_j})$, a valuation σ_{i_j}, and a fact $t_{i_j} \in v_{i_j}$, such that $t_{i_j} = \sigma_{i_j}(head(\varphi_{i_j}))$ and $\theta(b_i) \in refresh(\sigma_{i_j}(body(\varphi_{i_j})))$. If we take a query ψ such that $body(\psi) = \sigma_{1_j}(head(\varphi_{1_j})), \sigma_{2_j}(head(\varphi_{2_j})), \ldots, \sigma_{n_j}(head(\varphi_{n_j}))$, and $head(\psi) = (\sigma_{1_j} \cup \sigma_{2_j} \cup \cdots \cup \sigma_{n_j})(head(\varphi))$, we have a containment mapping (namely θ), from φ to ψ^{exp}. Consequently ψ will be an element in the union of queries $\tilde{\varphi}$, and obviously ψ generates the fact t when applied to $\mathcal{S}$. Now let us consider the case that a given variable appears somewhere else in $\hat{\varphi}(T(\mathcal{S}))$. Again there are two cases to consider, either the repeating variable appears in the same atom or it appears in a different atom. In both cases, the same variables are in $\hat{\varphi}(T(\mathcal{S}))$ because there were the same variables in $T(\mathcal{S})$ – remember that $\hat{\varphi}$ always substitutes variables from the query for the variables from the tableau. The same variables could appear in the tableau only if they come from inversion of the same tuple and, moreover, from the same existential variable in the query defining view. And since this is the case it is obvious that in the corresponding atoms in $\tilde{\varphi}$ there will be the same terms in the corresponding positions. ■

Example 8 For example, let the source collection be $\mathcal{S} = \{(V_1(X_1, X_3) \leftarrow R(X_1, X_2), S(X_2, X_3), \{V_1(a, c)\}), (V_2(X_1) \leftarrow T(X_1, X_2), \{V_2(c)\})\}$, and let the query φ be $Q(X, Y, Y, Z, W) \leftarrow R(X, Y), S(Y, Z), T(Z, W)$. In this case there is one (minimal) p-contained rewriting of φ, namely $\psi = Q(X, Y) \leftarrow V_1(X, Y), V_2(Y)$. Applying ψ_φ gives us the atom $Q(a, f_{1,2}(a), f_{1,2}(a), c, f_{2,2}(c))$, and applying the *replace* function yields the answer $\{Q(a, Y, Y, c, W)\}$. Had we used the tableau method instead, we would have gotten $T(\mathcal{S}) = \{R(a, Y), S(Y, c), T(c, W)\}$. Then applying $\hat{\varphi}$ to $T(\mathcal{S})$

would have given $\{Q(a, Y, Y, c, W)\}$.

Example 9 For an additional example, consider the following source collection $\mathcal{S} = \{(V_1(X_1, X_3) \leftarrow R(X_1, X_2, X_3), \{V_1(a, b)\}), (V_2(X_1, X_2) \leftarrow S(X_1, X_2), \{V_2(c, a)\}), (V_3(X_2) \leftarrow S(X_1, X_2), \{V_3(a)\})\}$, and let φ be $Q(X, Y, Z, W) \leftarrow S(X, Y), R(Y, Z, W)$. The two (minimal) p-contained rewritings of φ: $Q(X, Y, W) \leftarrow V_2(X, Y), V_1(Y, W)$ and $Q(Y, W) \leftarrow V_3(Y), V_1(Y, W)$. Applying ψ_φ gives us two atoms $Q(c, a, , f_{1,2}(a, b), b)$ and $Q(f_{3,1}(a), a, f_{1,2}(a, b), b)$, and applying the *replace* function yields the answer $\{Q(c, a, Z, b), Q(X, a, Z, b)\}$. Had we constructed the tableau for this source collection first, we would have gotten $T(\mathcal{S}) = \{R(a, Z, b), S(c, a), S(X, a)\}$. Then applying $\widehat{\varphi}$ to $T(\mathcal{S})$ would have given $\{Q(c, a, Z, b), Q(X, a, Z, b)\}$.

Note that since $\widetilde{\varphi}(\mathcal{S})$ computes a tableau that is equivalent to $\widehat{\varphi}(T(\mathcal{S}))$ the result of this computation can be used for subsequent querying, which was not possible with the evaluation that we presented in our earlier work [GK02].

In the next section we give an algorithm that, for a given conjunctive query φ, computes a finite union of conjunctive queries equivalent to $\widehat{\varphi}$, and a modification of it that encodes into the queries the containment mappings μ needed in the evaluation.

3.2 Query rewriting

Since we generalized the notion of containment mapping to p-containment mapping it is only natural that any rewriting algorithms (such as the algorithms in [LRO96, PL00, ALM02]), which are based on containment mappings, can be extended to produce the p-rewriting. But first we have to look at complete characterization and complexity of the problem of finding maximally contained p-rewriting of given query using views.

3.2.1 Complexity of finding p-rewriting

The issue of complexity of query containment and of finding rewriting of a given query using views has been extensively studied in the literature [LMSS95, AD98, CGLV00].

Let us first restate some fundamental results shown in [LMSS95]. Note that the statements were slightly modified in order to retain consistency in the notions used in this thesis.

Lemma 4 *Let φ and ψ be conjunctive queries. There is a rewriting of φ using ψ if and only if $\pi_\emptyset(\varphi) \subseteq \pi_\emptyset(\psi)$, i.e. , the projection of φ onto the empty set of columns is contained in the projection of ψ onto the empty set of columns.* ■

The statement of the above Lemma is equivalent to the following statement: If ψ is empty for a given database, then so is φ.

Theorem 5 *Let $\mathcal{S}$ be a source collection and φ be a conjunctive query over $sch(\mathcal{S})$. If ψ is a contained rewriting of φ using $\mathcal{S}$, then there exist a contained rewriting χ such that $\psi \subseteq \chi$ and number of atoms in $body(\chi)$ is not greater than number of atoms in $body(\varphi)$.* ■

Now using the above theorem of [LMSS95] and the definition of p-rewriting we can easily show the following.

Corollary 2 *Let $\mathcal{S}$ be a source collection and φ be a conjunctive query over sch($\mathcal{S}$). If ψ is a p-contained rewriting of φ using $\mathcal{S}$, then there exist a p-contained rewriting χ such that*

- $\psi \subseteq \chi$
- *number of atoms in body(χ) is not greater than number of atoms in body(φ).*

■

In [LMSS95], using Lemma 4, Theorem 5 and earlier results on the complexity of the containment [CM77, vdM92], the authors show the following complexity results.

Theorem 6 *Let $\mathcal{S}$ be a source collection and φ be a conjunctive query over sch($\mathcal{S}$). The problem of determining whether there exists a contained rewriting of φ using $\mathcal{S}$ is NP-complete.* ■

Given the fact that a rewriting is a special case of a p-rewriting, we can now show that shifting from classical rewritings to p-rewritings does not affect the computational complexity.

Theorem 7 *Let $\mathcal{S}$ be a source collection and φ be a conjunctive query over sch($\mathcal{S}$). The problem of determining whether there exist a p-contained rewriting of φ using $\mathcal{S}$ is NP-complete.*

Proof.

Given the fact that the problem of existence of a contained rewriting of φ using $\mathcal{S}$ is a special case of existence of a p-contained rewriting of φ using $\mathcal{S}$, NP-hardness follows from the Theorem 6.

The problem is in NP because, by the Corollary 2, we need only consider p-rewritings with the body that has no more atoms than the *body*(φ). We can guess such a p-rewriting, guess the containment mapping from φ to the p-rewriting, and verify the correctness of our guesses.

■

The Theorem 7 suggests an easy algorithm for finding p-rewritings for a given query φ. All we need to do is to consider all queries ψ such that *body*(ψ) contains as many atoms of *desc*$(\mathcal{S})$ as the number of atoms in φ and *head*$(\psi) = \pi_L(\varphi)$. And for each such ψ we need to test whether there is a p-containment mapping from φ to ψ^{ext}. Unfortunately, the number of possible p-rewritings is exponential in the size of the query. In the remainder of this chapter we will give two algorithms that use structure of the query to reduce the number of query rewritings that need to be considered.

3.2.2 P-bucket and sp-bucket algorithms

The following algorithm is a generalization of the bucket algorithm [LRO96] and, therefore, we call it the *p-bucket algorithm*. The main idea underlying the bucket algorithm is that the number of query rewritings that need to be considered can be reduced if we first examine each atom in the query in isolation, and determine which view may be relevant to this atom.

In order to formalize the p-bucket algorithm we need a few concepts. A *unifier* for two atoms a and b is a substitution θ such that $\theta(a) = \theta(b)$. A substitution θ is *more general* than a substitution ζ if for some substitution ζ', $\zeta = \theta \circ \zeta'$. A *most general unifier* for a and b is a unifier θ such that, for each unifier ζ of a and b, θ is more general than ζ.

Given a query φ the p-bucket algorithm proceeds in two steps.

In the first step, the algorithm creates a bucket for each atom in φ. Then the buckets are populated by source atoms that are relevant to answering the particular atom. More specifically, consider a bucket for an atom b_φ of φ, and a source $S_i = (\varphi_i, v_i)$. If $body(\varphi)$ contains an atom b_{φ_i} such that there is a (most general) unifier θ for b_φ and b_{φ_i}, then $\theta(head(\varphi_i))$ is put in the bucket of atom b_φ. In case the atom b_φ unifies with more than one atom in a source S_i the bucket of b_φ will contain multiple occurrences of unified $head(\varphi_i)$.

In the second step, the algorithm considers query rewritings that are conjunctive queries, each consisting of one conjunct from every bucket. The head of each rewriting is the projection of variables that are in the body of this rewriting. For each rewriting, the algorithm checks whether it's expansion is p-contained in the query. If so, the rewriting is added to the answer. Though not required, a check can be added to determine that the resulting rewriting is not redundant. Hence, the result of algorithm is a union of conjunctive rewritings.

Theorem 8 *The union of all rewritings produced by the p-bucket algorithm relative to a query φ is equivalent to the union of all p-contained rewritings of φ.*

Proof.

The p-bucket algorithm produces *only* semantically correct rewritings since it tests for p-containment of each of candidate solutions.

For the proof that the output of the algorithm contains *all* semantically correct rewritings we have to prove that if there exists a p-rewriting ψ of a given conjunctive query φ then there will be a p-rewriting χ in the output of the p-bucket algorithm, such that $\psi \subseteq_p \chi$.

It has been known since Chandra and Merlin's paper [CM77] that every conjunctive query has a unique (up to renaming of variables) minimal equivalent query that can be obtained by deletion of zero or more atoms. Let us call the minimal equivalent of ψ ψ_{min} and the minimal equivalent of χ χ_{min}. Since ψ is a p-rewriting of φ $\psi^{exp} \subseteq_p \varphi$ and consequently $(\psi_{min})^{exp} \subseteq_p \varphi$. It is easy to see that ψ_{min} cannot have more atoms than φ because each atom of ψ_{min} covers at least one atom of φ. We now have two cases: either ψ_{min} has the same number of atoms as φ, or ψ_{min} has fewer atoms then φ.

If ψ_{min} has the same number of atoms as φ then each atom of ψ_{min} would be placed in the corresponding bucket by the first phase of p-bucket algorithm. In the second phase the algorithm produces cross-product of the contents of all buckets and, thus, it would produce χ that has all of ψ_{min} atoms. Then the algorithm would compute the $head(\chi)$ that would be the longest possible list of variables of the $head(\varphi)$. Therefore, the p-bucket algorithm would produce χ such that $\psi_{min} \subseteq_p \chi$.

If ψ_{min} has fewer atoms than φ then each atom of ψ_{min} would be placed in the corresponding bucket by the first phase of p-bucket algorithm. In the second phase the algorithm produces cross-product of the contents of all buckets and, thus, it would produce χ that has all of ψ_{min} atoms plus some redundant atoms. Then the algorithm would compute the $head(\chi)$ that would be the longest possible list of variables of the $head(\varphi)$. Therefore, the p-bucket algorithm would produce χ such that $\psi_{min} \subseteq_p \chi_{min}$.

In [SY80] authors show that to test the containment of unions of conjunctive queries $U_1 \subseteq U_2$ is sufficient to show that for each query $\varphi_1 \in U_1$ there exists a query $\varphi_2 \in U_2$ such that $\varphi_1 \subseteq \varphi_2$. Thus, the equivalence of the claim of the theorem follows.

■

Example 10 We illustrate the algorithm with the Example 2 given in the introduction. The first step of the algorithm will construct and populate two buckets, one for each of the atoms in the query. The contents of the buckets are shown in Figure 4.

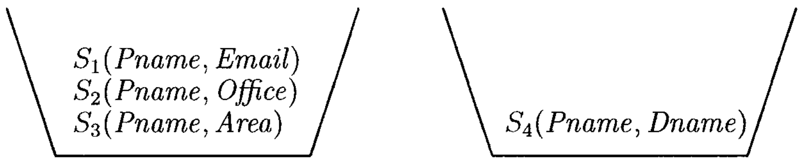

Figure 4: Contents of the buckets in the first step of the p-bucket algorithm for the query in the Example 2.

The second step of the algorithm produces the following p-rewritings:

$$Q(Prof, Email) \leftarrow S_1(Prof, Email), S_4(Prof, compsci)$$

$$Q(Prof, Office) \leftarrow S_2(Prof, Office), S_4(Prof, compsci)$$

$$Q(Prof, Area) \leftarrow S_3(Prof, Area), S_4(Prof, compsci)$$

Obviously it might be desirable to compute the rewriting that encodes the containment mappings μ needed in the $\widetilde{\varphi}$ evaluation in the rewriting directly, rather than recompute them at the evaluation time.

To do that we can modify our p-bucket algorithm as follows. First step of the algorithm remains unchanged. In the second step if expansion of the proposed rewriting is p-contained in the query the algorithm outputs one rewriting with skolemized head for each p-containment mapping from the query to the rewriting. We call this modified algorithm the *sp-bucket algorithm.*

The union of all rewritings produced by sp-bucket algorithm can then be evaluated on the source collection in the usual manner and *replace* function can be applied to the resulting set.

The following lemma directly follows from Theorem 8 and the construction of skolemized rewriting by sp-bucket algorithm.

Lemma 5 *The result of the evaluation of the union of all rewritings produced by the sp-bucket algorithm relative to a query* φ *on any source collection is equivalent to* $\widetilde{\varphi}$ *evaluation of* φ.

Proof.

It is obvious that the union of all rewritings produced by sp-bucket algorithm contains one skolemized rewriting for each p-contained rewriting produced by p-bucket algorithm. The claim follows from the Theorem 8 and the fact that function terms are inserted by sp-bucket algorithm in the same way as by $\widetilde{\varphi}$. ■

Example 11 For the previous example sp-bucket algorithm produces the following skolemized p-rewritings:

$$Q(Prof, Email, f_{1,3}(Prof, Email), f_{1,4}(Prof, Email)) \leftarrow S_1(Prof, Email), S_4(Prof, compsci)$$

$$Q(Prof, f_{2,2}(Prof, Office), Office, f_{2,4}(Prof, Office)) \leftarrow S_2(Prof, Office), S_4(Prof, compsci)$$

$$Q(Prof, f_{3,2}(Prof, Area), f_{3,3}(Prof, Area), Area) \leftarrow S_3(Prof, Area), S_4(Prof, compsci)$$

Assume that the source collection $\mathcal{S}$ has the following extension.
$ext(\mathcal{S}) = \{\ S_1(Murphy, murphy@cs.uoft.edu),\ S_1(Jones, jones@cs.concordia.ca),$
$S_2(Murphy, SF322),\ S_3(Smith, DB),\ S_3(Brown, AI),\ S_4(Murphy, compsci),$
$S_4(Jones, compsci),\ S_4(Black, elec),\ S_4(Smith, compsci),\ S_4(Brown, compsci)\}.$
The result of the $\widetilde{\varphi}$ evaluation of rewritings produced by sp-bucket algorithm on $\mathcal{S}$ corresponds to the table with nulls that is given in the introduction and reproduced in the Figure 5.

Pname	Email	Office	Area
Murphy	murphy@cs.uoft.edu	$\perp$	$\perp$
Murphy	$\perp$	SF322	$\perp$
Smith	$\perp$	$\perp$	DB
Jones	jones@cs.concordia.ca	$\perp$	$\perp$
Brown	$\perp$	$\perp$	AI

Figure 5: Table with nulls corresponding to the $\tilde{\varphi}$ evaluation of the rewritings in the Example 11.

3.2.3 P-MiniCon and sp-MiniCon algorithms

In the previous section we have used a generalization of the bucket algorithm for computation of the p-contained rewriting because of it simplicity, however, the combination step of bucket algorithm has several deficiencies and does not scale up well. The main inefficiency of the algorithm is that it misses some important interactions between view atoms by considering each atom in isolation. As a result, the buckets contain irrelevant views, and hence the second step of the algorithm becomes very expensive. The much more efficient unification based method was proposed by Grahne and Mendelzon in [GM99] and Pottinger and Halevy in [PL00] presented detailed and practical implementation of a unification based algorithm - the MiniCon algorithm.

The MiniCon begins like the bucket algorithm, considering which views contain atoms that correspond to atoms in the query. However, once the algorithm finds a partial mapping from a atom in the query to a atom in a view, it changes perspective and looks at the variables in the query. The MiniCon algorithm considers the joins in the query and finds the minimal additional set of atoms that need to be mapped to atoms in this view, preserving already discovered mappings. This set of atoms and mappings is called a MiniCon Description (MCD), and can be viewed as generalization of buckets. In the second phase the algorithm combines the MCDs to produce the rewriting, however, because of the way MCDs are constructed, containment checks are not needed in the second phase, giving MiniCon considerable speedup compared to the bucket algorithm.

Given that the MiniCon algorithm is based on the containment mappings, in the following we present a modification of the algorithm, which we call *p-MiniCon*, that produces p-contained rewriting. It should be noted that we also extended the original algorithm to allow a query to contain constants.

In what follows, given a query φ, we will denote variables of the query by $\mathbf{var}(\varphi)$ and constants of the query by $\mathbf{dom}(\varphi)$.

Formally, given a query φ and a source $S_i = (\psi, v)$ a MiniCon description (MCD) C is a tuple $(\eta_C, \psi_C, \theta_C, G_C)$, where:

- η_C is the least restrictive *head homomorphism* on a query ψ defined as a mapping from $\mathbf{var}(\psi) \cup \mathbf{dom}(\psi)$ to $\mathbf{var}(\psi) \cup \mathbf{dom}(\psi) \cup \mathbf{dom}(\varphi)$ that is the identity on the constants and existential variables but it may equate distinguished variables and/or map distinguished variable to a constant in either $\mathbf{dom}(\psi)$ or $\mathbf{dom}(\varphi)$, i.e., for every distinguished variable X, $\eta(X)$ is distinguished, and $\eta(X) = \eta(\eta(X))$. η_C is least restrictive in that it only equates the distinguished variables or maps them to constants if it is necessary in order to unify atoms of φ with atoms of ψ.

- ψ_C is *head*$(\eta_C(\psi))$. Note that ψ_C is uniquely defined by the other components of C and, thus, is included only for convenience.

- θ_C is an extended partial mapping from $\mathbf{var}(\varphi) \cup \mathbf{dom}(\varphi)$ to $\eta_C(\psi)$. that is identity on the constants and on variables it is defined as follows. Given a

ψ_C	η_C	θ_C	G_C
$S_1(Prof, Email)$	$Prof/Prof$, $Email/Email$	$Prof/Prof$, $Email/Email$, $Office/X_1$, $Area/X_2$	1
$S_2(Prof, Office)$	$Prof/Prof$, $Office/Office$	$Prof/Prof$, $Email/X_3$, $Office/Office$, $Area/X_4$	1
$S_3(Prof, Area)$	$Prof/Prof$, $Area/Area$	$Prof/Prof$, $Email/X_5$, $Office/X_6$, $Area/Area$	1
$S_4(Prof, compsci)$	$Prof/Prof$, $Dept/compci$	$Prof/Prof$, $compci/compci$,	2

Figure 6: MCDs formed by the p-MiniCon algorithm for the Example 2.

variable X of φ the θ_C is set as

$$\theta_C(X) = \begin{cases} \eta_C(X), \text{ if } \eta_C(X) \text{ occurs in } \mathit{head}(\psi) \\ \text{distinct fresh variable, otherwise.} \end{cases}$$

Moreover, θ_C has to satisfy the following properties. *Covering property*: Variable X is in the domain of θ_C, if there is a subgoal G that includes X and all variables in G are in the domain of θ_C. *Join property*: if there is more than one atom in φ that includes X either $\theta_C(X)$ maps X to a distinguished variable in $\eta_C(\psi)$, or for every atom G' in φ that includes X, all variables in G are in the domain of θ_C.

- G_C is a minimal subset of the atoms in φ, for which θ_C satisfies the join property.

The MCDs constructed by p-MiniCon algorithm for our running example (Example 2) are given in the Figure 6.

ψ_C	η_C	θ_C	G_C
$V_2(C,D)$	C/C, D/D	X/C, Y/D	3
$V_3(F,F)$	F/F, H/F	X/F,Y/F	1, 2, 3

Figure 7: MCDs formed by the p-MiniCon algorithm for the Example 12.

For an additional case of the construction of MCDs consider Example 12. In this example, no fresh variables are generated by θ_C, but it demonstrates importance of the join property of θ_C.

Example 12 Let the source collection be $\mathcal{S} = \{(V_1(A) \leftarrow R(A,B), R(B,A), \{\ldots\}),$ $(V_2(C,D) \leftarrow S(C,D), \{\ldots\})$, $(V_3(F,H) \leftarrow R(F,G), R(G,H), S(F,G)\{\ldots\})\}$, and let the query be $Q(X) \leftarrow R(X,Y), R(Y,X), S(X,Y)$. The MCDs formed for Q by p-MiniCon algorithm are shown in Figure 7 Note that no MCD is created for V_1 because doing so would violate join property of θ_C. This is due to the fact that B is existential variable in V_1 but S is not in $sch(V_1)$.

In the second phase p-MiniCon algorithm considers combinations of MCDs and for each valid combination creates a rewriting of the query. Given the method for constructing MCDs, the only valid combinations the algorithm needs to consider are those, where $G_{C_1} \cup \ldots \cup G_{C_n} = body(\varphi)$, and for every $i \neq j$, $G_{C_i} \cap G_{C_j} = \emptyset$. For every such combination of the MCDs $C_1, \ldots, C_n$, the algorithm produces a rewriting χ, where $body(\chi) = \psi_{C_1} \cup \ldots \cup \psi_{C_n}$ and for every variable Y in ψ_{C_i}, if $Y = \theta_{C_i}(X)$, Y is replaced by X, otherwise Y is replaced by a fresh variable; $\mathbf{dom}(head(\chi)) = \mathbf{dom}(head(\varphi))$ and $\mathbf{var}(head(\chi)) = \mathbf{var}(head(\varphi)) \cap \mathbf{var}(body(\chi))$. In other words, the algorithm replaces variables in the atoms of the body of the rewriting by variables of the query where possible, and head of the rewriting is the projection of those attributes of the query that are present in the body of the rewriting.

Example 13 Consider the MCDs constructed for our main example that are given in the Figure 6. The second step of the p-MiniCon algorithm produces the following p-rewritings:

$$Q(Prof, Email) \leftarrow S_1(Prof, Email), S_4(Prof, compsci)$$

$$Q(Prof, Office) \leftarrow S_2(Prof, Office), S_4(Prof, compsci)$$

$$Q(Prof, Area) \leftarrow S_3(Prof, Area), S_4(Prof, compsci)$$

Example 14 For the MCDs constructed for Example 12 that are given in the Figure 7 the p-MiniCon algorithm outputs only one p-rewriting: $Q(X) \leftarrow V_3(X, X)$.

It should be noted that after the combination step the rewriting may still contain redundant atoms. Removing them involves several tests for query containment and, as in case of p-bucket algorithm, is not essential for neither correctness nor completeness of the algorithm.

Theorem 9 *The union of all rewritings produced by the p-MiniCon algorithm relative to a query φ is equivalent to the union of all p-contained rewritings of φ.*

Proof.

Let φ be a query and let $\mathcal{S} = \{S_1(\psi_1, v_1), \ldots, S_n(\psi_n, v_n)\}$ be a source collection.

To prove that p-MiniCon outputs *only* semantically correct rewritings we need to show that there is a p-containment mapping from a query φ to the expansion of the p-rewriting χ, (χ^{exp}), produced by the p-MiniCon algorithm.

First, let us note how χ^{exp} is obtained from χ. For every atom H_i in χ there is an MCD C_i and G_{C_i} is the set of atoms of φ such that all variables of G_{C_i} are in the domain of θ_{C_i}. Now, for every atom G of φ, which is in G_{C_i} there will be an atom

$\tau(G)$ in χ^{exp}, where, τ is identity on constants and, for a variable X, $\tau(X) = X$ if X is in H, otherwise $\tau(X) = Y$, where Y is a fresh distinct variable.

Now we claim that τ is a p-containment mapping from φ to χ^{exp}.

To be a p-containment mapping τ has to satisfy two properties: (1) $\tau(\mathit{body}(\varphi)) \subseteq \mathit{body}(\chi^{exp})$, and (2) for every variable X in $\mathit{head}(\chi^{exp})$ there is a variable Y in $\mathit{head}(\varphi)$, such that $\tau(Y) = X$.

To see that (1) holds note the following. Every atom of $\mathit{body}(\varphi)$ is in some G_i, because second step of p-MiniCon to produce a rewriting used MCDs $C_1, \ldots, C_n$, such that $G_{C_1} \cup \ldots \cup G_{C_n} = \mathit{body}(\varphi)$. Every $X \in \mathbf{var}(\mathit{body}(\varphi))$ is in the domain of τ because in the second step of the algorithm replaced variables χ by the variables of of φ and because of covering property of θ_{C_i} enforced by the algorithm. τ is a one to one mapping because of the join property of θ_{C_i} enforced by the algorithm and, because at the combination step the algorithm used MCDs $C_1, \ldots, C_n$, such that for every $i \neq j$, $G_{C_i} \cap G_{C_j} = \emptyset$.

It is also easy to see that (2) is satisfied by τ, since χ was constructed so that $\mathbf{dom}(\mathit{head}(\chi)) = \mathbf{dom}(\mathit{head}(\varphi))$ and $\mathbf{var}(\mathit{head}(\chi)) = \mathbf{var}(\mathit{head}(\varphi)) \cap \mathbf{var}(\mathit{body}(\chi))$.

To prove that output of the algorithm contains *all* semantically correct rewritings we have we have to prove that if there exists a p-rewriting χ of a given conjunctive query φ then there will be a p-rewriting χ' in the output of the p-MiniCon algorithm, such that $\chi \subseteq_p \chi'$. The claim of the theorem will then follow from the characterization of containment of unions of conjunctive queries given in [SY80].

From [CM77] follows that every conjunctive query has a unique (up to renaming of variables) minimal equivalent query that can be obtained by deletion of zero or more atoms. Without the loss of generality let χ be such a minimal query. Since χ is a p-rewriting of φ $\chi^{exp} \subseteq_p \varphi$ and, consequently, there is a containment mapping μ from φ to χ^{exp}. We will use μ to show that there exists a set of MCDs that are created by p-MiniCon algorithm such that when the MCDs are combined by the algorithm, we obtain a p-rewriting χ' such that $\chi \subseteq_p \chi'$.

For each atom H_i of χ, we define G_i to be the set of atoms $G \in \varphi$, such that $\mu(G)$ is in expansion of H_i, i.e., G_i includes the set of atoms in φ that are mapped by μ to the expansion of G_i in χ^{exp}. Note that for $i \neq j$, the sets G_i and G_j are disjoined.

We denote by μ_i the restriction of containment mapping μ to the variables and constants appearing in G_i. That means μ_i is the mapping form $\mathbf{var}(G_i) \cup \mathbf{dom}(G_i)$ to $\mathbf{var}(H_i^{exp}) \cup \mathbf{dom}(H_i^{exp})$. However, μ_i can be rewritten as a composition of two mappings, one from $\mathbf{var}(G_i) \cup \mathbf{dom}(G_i)$ to $\eta_i(\mathbf{var}(\psi_i) \cup \mathbf{dom}(\psi_i))$ (where η_i is head homomorphism on ψ_i), and another from $\eta_i(\mathbf{var}(\psi_i) \cup \mathbf{dom}(\psi_i))$ to $\mathbf{var}(H_i^{exp}) \cup \mathbf{dom}(H_i^{exp})$. Formally, there exist a mapping from $\mathbf{var}(G_i) \cup \mathbf{dom}(G_i)$ to $\eta_i(\mathbf{var}(\psi_i) \cup \mathbf{dom}(\psi_i))$ and a renaming τ of the variables in $\eta(\psi_i)$, such that, for every variable $X \in G_i$, $\mu_i(X) = \tau(\theta_i(\eta_i(X)))$. We choose η_i to be the least restrictive head homomorphism on $\mathbf{var}(\psi_i) \cup \mathbf{dom}(\psi_i)$ for which θ_i and τ exist.

We know have all the components of a MCD, which we denote by C_i:

- η_{C_i} is the least restrictive head homomorphism on ψ_i.

- ψ_{C_i} is $head(\eta_{C_i}(\psi))$.

- θ_{C_i} is an extended partial mapping from $\mathbf{var}(\varphi) \cup \mathbf{dom}(\varphi)$ to $\eta_{C_i}(\psi)$. Note that, since μ_i is the restriction of a containment mapping, θ_{C_i} satisfies covering and join properties.

- G_{C_i}, however, is not guaranteed to be minimal. If it is the case that G_{C_i} is not minimal, we can fix it by decomposing C_i into a set of MCDs such that each of them covers exactly minimal set of atoms that satisfy the join property.

Now we have a set of MCDs $C_1, \dots C_k$ and it should be noted that it satisfies the following properties. First of all, because μ is a containment mapping from φ to $(\chi_{min})^{exp}$ $G_{C_1} \cup \ldots \cup G_{C_n} = body(\varphi)$. Secondly, because of the way we constructed G_i, for every $i \neq j$, $G_{C_i} \cap G_{C_j} = \emptyset$. As the consequence, at the combination step MiniCon algorithm will produce a rewriting χ' by combining MCDs $C_1 \ldots C_k$ with $\mathbf{dom}(head(\chi)) = \mathbf{dom}(head(\varphi))$ and $\mathbf{var}(head(\chi)') = \mathbf{var}(head(\varphi)) \cap \mathbf{var}(body(\chi'))$. Furthermore, since the renaming at this step is done consistently for all occurrences of the variables, there will be a containment mapping form χ' to χ, and, thus, $\chi \subseteq_p \chi'$.

■

As it was the case with p-bucket algorithm we can modify our p-MiniCon algorithm to encode in the rewriting the containment mappings μ needed in the $\tilde{\varphi}$ evaluation. We call the modified algorithm *sp-MiniCon* and the modifications are as follows. In the first step of the algorithm, given a query φ and a source $S_i = (\psi, v)$ a MiniCon description (MCD) C is a tuple $(\eta_C, \psi_C, \theta_C, G_C)$, where η_C, ψ_C, and G_C remain unchanged, but θ_C is an extended partial mapping from $\mathbf{var}(\varphi) \cup \mathbf{dom}(\varphi)$ to $\eta_C(\psi)$ that is identity on the constants and on variables it is defined as follows. Suppose a variable X of φ is is mapped by θ_C to a variable in an atom in $body(\psi)$, $\theta_C(X)$ is the j:th distinguished variable in the $body(\psi)$, and existential variables in ψ are $X_1, \ldots, X_k$. Then the θ_C is set as

$$\theta_C(X) = \begin{cases} \eta_C(X), \text{ if } \eta_C(X) \text{ occurs in } head(\psi) \\ f_{i,j}(\eta_C(X_1), \ldots, \eta_C(X_k)), \text{otherwise.} \end{cases}$$

The covering and join properties of θ_C remain unchanged.

Second step of the algorithm does not change except that, care has to be taken to preserve function terms and to propagate them into corresponding positions in the head of the rewriting.

ψ_C	η_C	θ_C	G_C
$S_1(Prof, Email)$	$Prof/Prof$, $Email/Email$	$Prof/Prof$, $Email/Email$, $Office/f_{1,3}(Prof, Email)$, $Area/f_{1,4}(Prof, Email)$	1
$S_2(Prof, Office)$	$Prof/Prof$, $Office/Office$	$Prof/Prof$, $Email/f_{1,3}(Prof, Office)$, $Office/Office$, $Area/f_{2,4}(Prof, Office)$	1
$S_3(Prof, Area)$	$Prof/Prof$, $Area/Area$	$Prof/Prof$, $Email/f_{3,2}(Prof, Area)$, $Office/f_{3,3}(Prof, Area)$, $Area/Area$	1
$S_4(Prof, compsci)$	$Prof/Prof$, $Dept/compci$	$Prof/Prof$, $compci/compci$,	2

Figure 8: MCDs formed by the sp-MiniCon algorithm for the Example 15.

Example 15 The MCDs constructed by sp-MiniCon for our main example (Example 2) are given in the Figure 8. Output of the second step of the algorithm is, indeed, the same as that of the sp-bucket (See Example 11):

$$Q(Prof, Email, f_{1,3}(Prof, Email), f_{1,4}(Prof, Email)) \leftarrow S_1(Prof, Email), S_4(Prof, compsci)$$

$$Q(Prof, f_{2,2}(Prof, Office), Office, f_{2,4}(Prof, Office)) \leftarrow S_2(Prof, Office), S_4(Prof, compsci)$$

$$Q(Prof, f_{3,2}(Prof, Area), f_{3,3}(Prof, Area), Area) \leftarrow S_3(Prof, Area), S_4(Prof, compsci)$$

Lemma 6 *The result of the evaluation of the union of all rewritings produced by the sp-MiniCon algorithm relative to a query φ on any source collection is equivalent to $\tilde{\varphi}$ evaluation of φ.*

Proof.

It is obvious that the union of all rewritings produced by sp-MiniCon algorithm contains one skolemized rewriting for each p-contained rewriting produced by p-MiniCon algorithm. The claim follows from the Theorem 9 and the fact that function terms are inserted by sp-MiniCon algorithm in the same way as by $\tilde{\varphi}$. ■

In the next chapter we give the pseudo-code of both sp-bucket and sp-MiniCon algorithms as implemented in our experimental system.

It should be noted that the asymptotic worst-case running time of the sp-MiniCon algorithm is the same as that of the sp-bucket algorithm. However, as can be seen form the results of experiments, that are presented in the next chapter, sp-MiniCon performs better in the average-case and scales up to a large number of views much better than sp-bucket algorithm.

Chapter 4

Experimental Results

4.1 Overview of the experiments

The goal of our experiments was twofold. The first goal was to compare the performance of the two proposed algorithms the sp-bucket and the sp-MiniCon in different circumstances. To do that we, first, fixed the number of sources and studied the effect of increasing the number of atoms in the body of the query and the arity of the output. Then, we fixed the arity of the output and number of atoms in the body of the query and measured performance of the algorithms as the number of sources in the source collection grew.

Secondly, we sought to validate the fact that our modifications to the original bucket and the MiniCon algorithms did not affect their performance. Therefore, we created test runs that simulated the experiments presented in [PL00] and considered two classes of queries and sources: chain and star. A chain query is a query where first atom is joined (has common variable(s)) with second, second with third, and so on. In star queries, there exists a unique atom in the query that is joined with every other atom and there are no joins between the other atoms.

To facilitate the experiments, we implemented the following modules in C++:

- *Sp-bucket algorithm.*

  The pseudo code of the algorithm is given in the Figure 4.1.

- *Sp-MiniCon algorithm.*

  The pseudo code of the algorithm is given in the Figure 4.1.

- *Random query generator.*

  The random query generator allows us to control the following parameters:

  - Number of atoms in the body of the query.
  - Arity of the query.
  - Shape of the query – chain, star, or random.

- *Random source collection generator.*

  The random source collection allows us to control the following parameters:

  - Number of sources in the collection.
  - Properties of the queries defining the sources:
    - Number of atoms in the body of the query.
    - Arity of the query.
    - Shape of the query – chain, star, or random.

Our experiments were all run on a computer with 450MHz Intel Pentium II processor and 512MB RAM running Windows 2000.

```
/* $\varphi$ is a query and $\mathcal{S} = \{S_1(\psi_1, v_1), \ldots, S_n(\psi_n, v_n)\}$ is a source collection */

          For each atom $g_i \in \varphi$
             Create a new bucket $b_i$
                For each $\psi_j$
                   For each atom $h_k \in \psi_j$
                      If $g_i$ unifies with $h_k$
                      Then
                         Find MGU $\theta$ for $g_i$ and $h_k$
                         Put $\theta(\psi_j)$ in $b_i$

          Create empty union of sp-rewritings $\Phi$

          If $b_1 \neq \emptyset \wedge \ldots \wedge b_n \neq \emptyset$
             Compute cross-product of buckets $\mathcal{X} = b_1 \times \ldots \times b_n$
             For each $X \in \mathcal{X}$
                If $\mu(\varphi) \subseteq (\nu(X))^{exp}$
                               for some substitution $\nu$ and some mapping $\mu$
                   Create empty sp-rewriting $\chi_l$
                   Set $body(\chi_l) = \nu(X)$
                   Set $head(\chi_l)$ as:
                      For every constant $a \in head(\varphi)$
                         Put $a$ in $head(\chi_l)$
                      For every variable $Y \in head(\varphi)$
                      [$\mu(Y) \in (\nu(\theta(\psi_j)))^{exp} \wedge \theta(\psi_j) \in X$]
                      [$Z_1, \ldots Z_k$ are distinguished in $\theta(\psi_j)$]
                      [$W_1, \ldots W_m$ are existential in $\theta(\psi_j)$]
                         If $\mu(Y) = Z_i$
                            Put $\nu(Z_i)$ in $head(\chi_l)$
                         Else $\mu(Y) = W_i$
                            Put $f_{j,i}(\nu(Z_1), \ldots, \nu(Z_k))$ in $head(\chi_l)$
                   Add $\chi_l$ to $\Phi$


          $\Phi$ is union of sp-rewritings of $\varphi$
```

Figure 9: Pseudo-code of the sp-bucket algorithm.

```
/* $\varphi$ is a query and $\mathcal{S} = \{S_1(\psi_1, v_1), \ldots, S_n(\psi_n, v_n)\}$ is a source collection */

          Create empty set of MCDs $\mathcal{C}$

          For each $\psi_i$
             For each atom $h_j \in \psi_i$ and each atom $g_k \in \varphi$
                If $\theta(g_k) = \eta(h_j)$
                      for some substitution $\theta$ and
                      (least restrictive) head homomorphism $\eta$
                   For each extension $\theta_{C_k}$ of $\theta$ and
                   each extension $\eta_{C_k}$ of $\eta$
                      Create MCD $C_k$
                      $C_k = (\eta_{C_k}, \psi_{C_k}, \theta_{C_k}, G_{C_k})$ as:
                      $\eta_{C_k}$ is (least restrictive) head homomorphism
                      $\psi_{C_k}$ is $head(\eta_{C_k}(\psi_i))$
                      $\theta_{C_k}$ (from $\mathbf{var}(\varphi) \cup \mathbf{dom}(\varphi)$ to $\eta_C(\psi_i)$)
                         $\theta_{C_k}$ is identity on the constants and
                         $\theta_{C_k}$ is defined on variables as
                            [Variable $X \in \varphi$]
                            [$\theta_{C_k}(X)$ is $j$:th distinguished variable in $body(\psi_i)$]
                            [$X_1, \ldots, X_k$ are existential variables in $\psi_i$]
                            $\theta_C(X) = \begin{cases} \eta_C(X), & \text{if } \eta_C(X) \text{ occurs in } head(\psi) \\ f_{i,j}(\eta_C(X_1), \ldots, \eta_C(X_k)), & \text{otherwise.} \end{cases}$
                      $G_{C_k}$ is minimal subset of atoms of $\varphi$, for which
                         $\theta_{C_l}$ satisfies covering and join properties
                      Add $C_k$ to $\mathcal{C}$

          Create empty sp-rewriting $\Phi$

          For every subset $C_1, \ldots, C_n$ of $\mathcal{C}$
             If $G_{C_1} \cup \ldots \cup G_{C_n} = body(\varphi)$ and, for every $i \neq j$, $G_{C_i} \cap G_{C_j} = \emptyset$
                   Create empty p-rewriting $\chi_l$
                   Set $body(\chi_l) = \psi_{C_1} \cup \ldots \cup \psi_{C_n}$
                   Set $head(\chi_l) = \bigcup_{i \in \{1, \ldots n\}} \theta_{C_i}(head(\varphi))$
                   For every variable $Y$ in $\chi_l$
                      If $Y = \theta_{C_i}(X)$
                         Replace $Y$ by $X$
                      Else
                         Replace $Y$ by fresh variable
                   Add $\chi_l$ to $\Phi$

          $\Phi$ is union of sp-rewritings of $\varphi$
```

Figure 10: Pseudo-code of the sp-MiniCon algorithm.

4.2 Summary of the results and discussion

For the first type of the experiments, where we sought to compare the performance of the two proposed algorithms, the sp-bucket and the sp-MiniCon, in different circumstances we have considered several different test runs.

First, we fixed the number of sources and studied the effect of increasing the number of atoms in the body of the query and the arity of the output. We created the test runs were the number of sources was fixed (10, 25, and 50) and considered cases where size of the body of the query ranged from 1 to 10 atoms, and arity of the query ranged from 1 to 10.

The increase in the number of atoms in the body of the query had a rather predictable effect on the running time of the algorithms. In case of the sp-bucket algorithm the time was rapidly growing as the number of atoms in the body of the query was growing, because the number of buckets was increasing and, most importantly, the number of expensive containment mapping checks done in the second step of the algorithm was increasing. In case of the sp-MiniCon algorithm the running time was also growing but at the different rate. The increase here is due to the increased time taken to construct MCDs but since there is no containment mapping involved at the combination step the growth rate was almost linear. In the Figure 11 we present the average results for 10 test runs with 10 sources in the source collection, queries of arity 2 and the number of atoms in the body of the query ranging form 1 to 10.

The variations in arity of the output had almost no impact on the running time of the algorithms and, thus, we omit the results of these experiments. This is due to the fact that, in case of the computation of the exact answer, number of rewritings in the output of any algorithm does not depend on the availability of the attributes in the body of the rewritings.

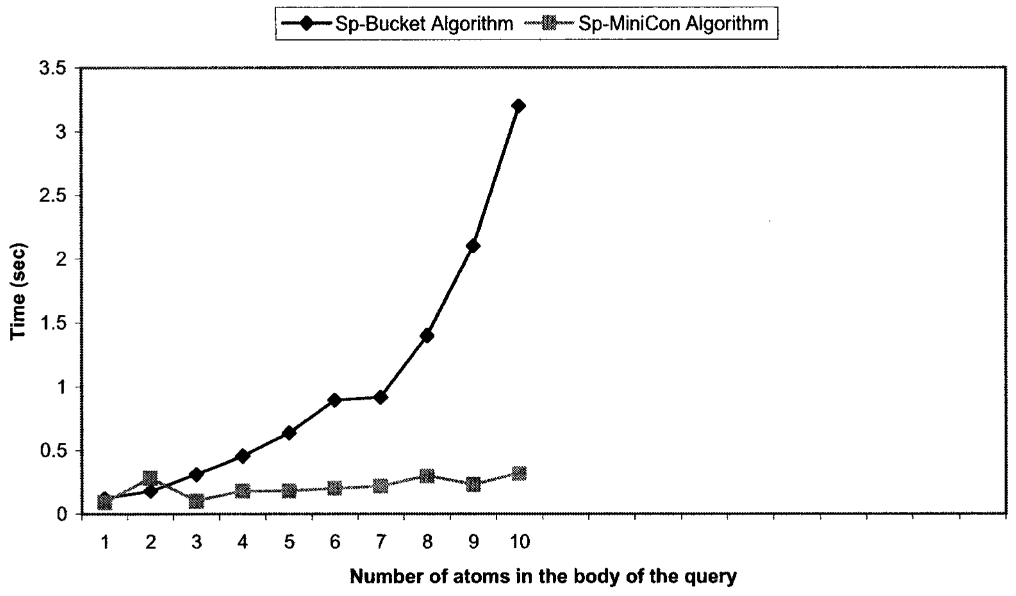

Figure 11: Average results of the test runs for the queries with number of atoms in the body of the query ranging from 1 to 10.

Second, we fixed the arity of the output (between 2 and 4) and number of atoms in the body of the query (between 3 and 5) and measured performance of the algorithms as the number of sources in the source collection grew. The source collection contained sources with randomly generated definitions with the number of atoms between 2 and 5. In the Figure 12 we present the average results for 10 test runs with number of sources ranging form 5 to 100. It can be seen that the sp-MiniCon outperforms the sp-bucket algorithm by the order of magnitude.

The other important factor affecting the running time of both algorithms was the arity of the queries defining the sources. In case where everything in the body of the source definition is accessible through the head of the source practically all combinations of sources produce a rewriting. Hence, any complete algorithm is forced to form a possibly exponential number of rewritings and demonstrates its worst-case performance. Since, the running time of the sp-MiniCon algorithm gets closer to

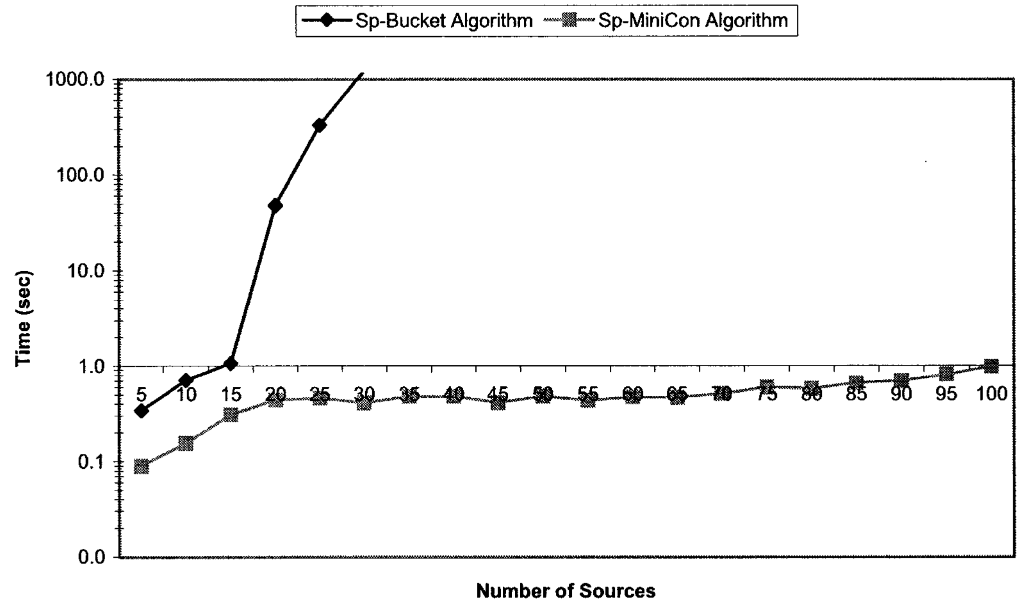

Figure 12: Average results of the test runs for the number of source ranging from 5 to 100.

the running time of the bucket algorithm. We omit the details of these experiments due to the fact that the experiment presented in Figure 13, which considered chain queries and source definitions with 5 atoms in the body and all variables distinguished demonstrate the point.

Secondly, we wanted to validate the fact that our modifications to the original bucket and the MiniCon algorithms did not affect their performance. For this reason we we created test runs that simulated the experiments presented in [PL00]. There are two experiments that report running times for both, the bucket and the MiniCon algorithms and in both of them the queries and the source definitions have the same shape and size. The first experiment considered chain queries and source definitions with 5 atoms in the body and all variables distinguished. The second experiment considered star queries and source definitions with 10 atoms in the body of the query with distinguished non-joined variables. The average results of 40 test runs of our

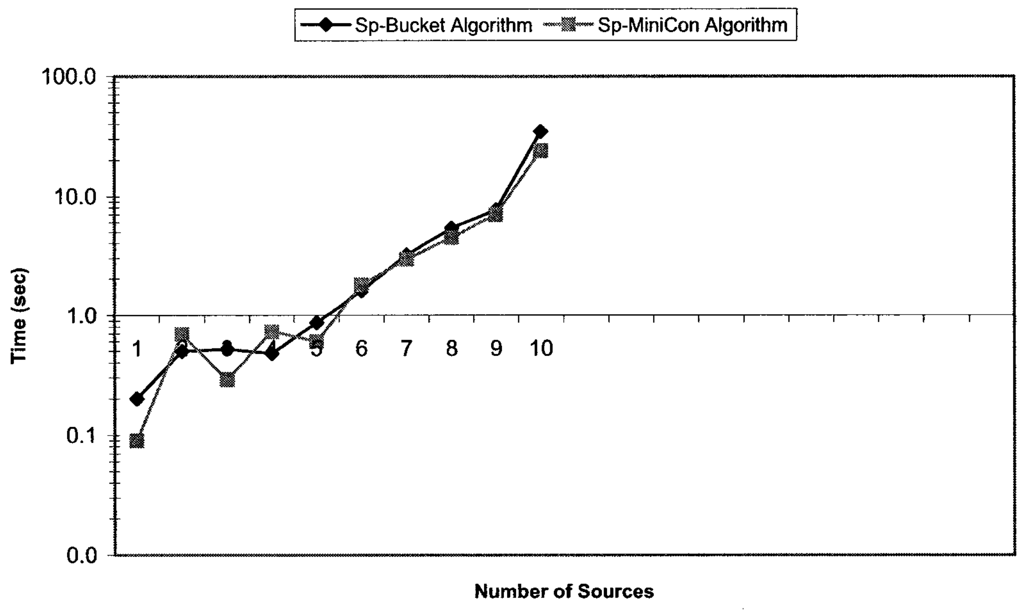

Figure 13: Average results of the test runs for the chain queries and source definitions.

experiments are presented in Figures 13 and 14, and it could be seen from that our results are comparable to the results of corresponding experiments in [PL00].

There is a difference in the runtime of the algorithms in our experiments of about 10 percent in the favor of the sp-bucket algorithm but this is reasonable given the different implementation. Nevertheless, Pottinger and Halevy also report the liner scalability of the MiniCon algorithm in the average case, and demonstrate the both algorithms perform the same in the worst case.

It sould be noted that the worst-case asymptotic running time of both the sp-bucket and the sp-MiniCon algorithms is $O(nmS)^n$, where n is the number of atoms in the query, m is the maximal number of atoms in a source, and S is the number of sources.

In summary, our experiments showed the following points. First, we have established that computation of the exact answer can be done efficiently for all practical situations including the large-scale systems. The sp-MiniCon algorithm scales up

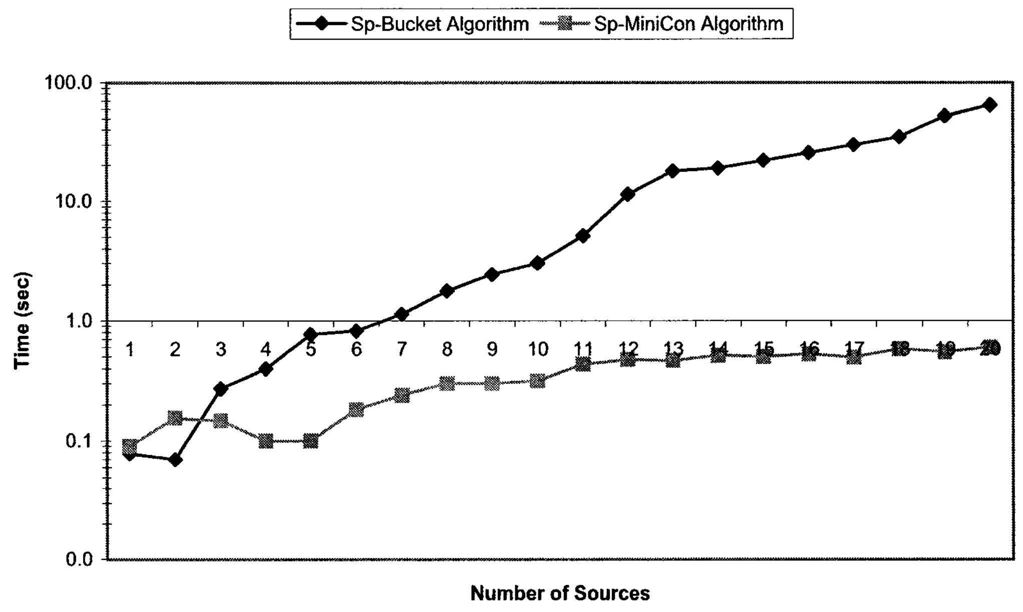

Figure 14: Average results of the test runs for the star queries and source definitions.

to large number of sources, significantly outperforms the sp-bucket algorithm in all considered average cases, and both algorithms demonstrate the same worst-case performance. Finally, we have confirmed that shifting from the computation of the certain answer to the computation of the exact answer does not change the average running time of query rewriting algorithms.

Chapter 5

Conclusions and Future Directions

5.1 Contributions

In the context of local-as-view information integration system a source collection defines a set of possible databases, therefore, querying a source collection, at least conceptually, means applying the query to each possible database, obtaining a set of possible answers. With this in mind we introduced the notion of exact answer, which can be represented as a relation containing null values, and we gave two methods for computing the exact answer as illustrated by the Figure 5.1.

The figure illustrates the fact that the path labeled $\varphi \circ \mathit{poss}$ commutes with the path labeled $\mathit{rep} \circ \widetilde{\varphi}$ and the path labeled $\mathit{rep} \circ \widehat{\varphi}$. Note that commutativity is with respect to coinitiality (see Theorems 2 and 4).

The first method involves explicitly computing $T(\mathcal{S})$ – a syntactic representation of the set of global databases implicitly defined by the sources. The $\widehat{\varphi}$ evaluation is then used to compute the exact answer. However, computing $\widehat{\varphi}(T(\mathcal{S}))$ might involve a lot of redundant work, since it amounts to constructing the tableau corresponding to the entire $\mathit{ext}(\mathcal{S})$, whereas the global relations that are in $\mathit{body}(\varphi)$ might be mentioned

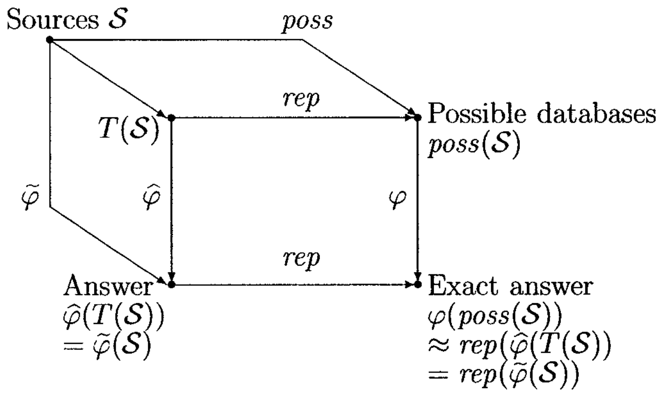
Sources $\mathcal{S}$
$poss$
rep
$T(\mathcal{S})$
Possible databases
$poss(\mathcal{S})$
$\tilde{\varphi}$
$\hat{\varphi}$
φ
rep
Answer
$\hat{\varphi}(T(\mathcal{S}))$
$= \tilde{\varphi}(\mathcal{S})$
Exact answer
$\varphi(poss(\mathcal{S}))$
$\approx rep(\hat{\varphi}(T(\mathcal{S}))$
$= rep(\tilde{\varphi}(\mathcal{S}))$

Figure 15: Semantics of a query in an information integration system.

in only few source definitions. Furthermore, the query might have selections and joins that could be computed directly at the sources.

The other method reformulates a query in terms of the source relations, and, thus, avoids inverting the entire source collection. In order to achieve this we generalized classical notion of query containment to p-containment and used this notion to reformulate the query as a union of p-contained conjunctive queries. Then we defined $\tilde{\varphi}$-evaluation that uses p-contained rewritings to compute the exact answer on the source collection.

We have also shown that shifting from classical rewritings to p-rewritings does not affect the computational complexity. Given the fact that a rewriting is a special case of a p-rewriting, the NP-hardness of testing whether a non-empty rewriting exists still holds. Likewise, our modification to the bucket-algorithm obviously doesn't increase its complexity; it remains computable in non-deterministic polynomial time. Computing the exact answer, either through the tableau or using a rewriting, remains in polynomial time.

As the result of our experiments we have established that computation of the exact answer can be done efficiently for all practical situations including the large-scale systems. Finally, we have confirmed that shifting from the computation of the certain answer to the computation of the exact answer does not change the average running time of query rewriting algorithms.

5.2 Future research directions

Query processing in information integration systems is significantly different than query processing in a traditional or a distributed database. The sources typically provide limited interfaces, partial but potentially overlapping data, they export semistructured data, and redundant and conflicting data are commonplace. Query optimization in the context of integrating heterogeneous data sources has lately received significant attention of the database community.

The notion of exact answer gives rise to many different optimizations. Even though rewriting, in general, outperforms tableau based query evaluation, tableau based techniques allows to use constraint information about the sources to minimize data transfer between the sources and information integration system, which could make this approach to work better in some situations. Other optimizations that we are considering include ordering of answers based on the number of values computed for the attributes in the tuples and allowing the user to specify that certain attributes are a must in the answer. These variations not only improve the usability of the system but also have significant impact on the performance. Furthermore, if we cache exact answers to some queries we can use them in subsequent query answering in order to reduce data transfer between the sources and the information integration system, which was not possible without the semantics of the exact answer. These and other optimization issues are currently under investigation.

Bibliography

[AD98] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *17th ACM Symp. on Principles of Database Systems (PODS)*, pages 254–265, Seattle, Washington, 1998.

[AGL01] A.Cali, G. De Giacomo, and M. Lenzerini. Models for information integration: Turning local-as-view into global-as-view. In *Int'l Workshop on Foundations of Models for Information Integration.*, Viterbo, Italy, 2001.

[ALM02] F. Afrati., C. Li, and P. Mitra. Answering queries using views with arithmetic comparisons. In *21st ACM Symp. on Principles of Database Systems (PODS)*, pages 209–220, Madison, Wisconsin, 2002.

[CGLV00] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi. What is query rewriting? (position paper). In *Cooperative Information Agents (CIA)*, pages 51–59, Boston, Massachusetts, 2000.

[CM77] A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries. In *9th ACM SIGACT Symp. on the Theory of Computing (STOC)*, pages 77–90, Boulder, Colorado, 1977.

[DG97] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *16th ACM Symp. on Principles of Database System (PODS)*, pages 109–116, Tuscon, Arizona, 1997.

[GK02] Grahne G. and V. Kiricenko. Obtaining more answers from information integration systems. In *5th Int'l Workshop on the Web and Databases (WebDB)*, pages 67–76, Madison, Wisconsin, 2002.

[GK03] Grahne G. and V. Kiricenko. Partial answers in information integration systems. In *5th Int'l Workshop on Web Information and Data Management(WIDM)*, New Orleans, Louisiana, 2003.

[GM99] G. Grahne and A. O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *7th Int'l Conference on Database Theory (ICDT)*, pages 332–347, Delphi, Greece, 1999.

[Gra91] G. Grahne. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, 1991.

[Hal01] A. Y. Halevy. Answering queries using views. *VLDB Journal*, 10:270–294, 2001.

[IL84] T. Imielinski and W. Lipski. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.

[Len02] M. Lenzerini. Data integration: A theoretical perspective. invited tutorial. In *21st ACM Symp. on Principles of Database Systems (PODS)*, pages 233–246, Madison, Wisconsin, 2002.

[LMSS95] A. Y. Levy, A. O. Mendelzon, Y. Sagiv, and D. Srivastava. Answering queries using views. In *14th ACM Symp. on Principles of Database Systems (PODS)*, pages 95–104, San Jose, California, 1995.

[LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *22nd Int'l Conf. on Very Large Databases (VLDB)*, pages 251–262, Mumbai (Bombay), India, 1996.

[Men84] A. O. Mendelzon. Database states and their tableaux. *ACM Trans. on Databases Systems*, 9(2):264–282, 1984.

[MM01] A. O. Mendelzon and G. Mihaila. Querying partially sound and complete data sources. In *20th ACM Symp. on Principles of Database Systems (PODS)*, pages 77–89, Santa Barbara, California, 2001.

[PL00] R. Pottinger and A. Y. Levy. A scalable algorithm for answering queries using views. In *26th Int'l Conference on Very Large Databases (VLDB)*, pages 484–495, Cairo, Egypt, 2000.

[Rei78] R. Reiter. On closed world databases. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 56–76, New York, New York, 1978. Plenum Press.

[SY80] Y. Sagiv and M. Yannakakis. Equivalence among relational expressions with the union and difference operators. *J. ACM*, 27(4):633–655, 1980.

[Ull97] J. D. Ullman. Information integration using logical views. In *6th Int'l Conference on Database Theory (ICDT)*, pages 19–40, Delphi, Greece, 1997.

[vdM92] R. van der Meyden. The complexity of querying indefinite data about linearly ordered domains. In *11th ACM Symp. on Principles of Database Systems (PODS)*, pages 331–345, XX, YY, 1992.