Query Equivalence and Containment on Relational and Tree Databases

Fang Wei

2004
Dissertation
zur Erlangung des Doktorgrades
der Fakultät für Angewandte Wissenschaften
der Albert-Ludwigs-Universität Freiburg im Breisgau

Dekan: Prof. Dr. Thomas Ottmann
Erstreferent: Prof. Dr. Georg Lausen
Zusammenfassung

Das „Query Containment” Problem handelt von der Überprüfung der Teilmengenrelation zweier Ergebnismengen die von zwei Anfragen an eine beliebige Datenbank resultieren.

Algorithmen zur Lösung des „Query Containment” Problems spielen eine wichtige Rolle innerhalb verschiedener Teilbereiche von Datenbanken.

Das „Query Containment” Problem kann im Wesentlichen unter den folgenden drei Kriterien betrachtet werden:

- die Anfragesprache,
- die Berücksichtigung von Constraints,
- die Struktur der Datenbank.

In der vorliegenden Arbeit werden unter Berücksichtigung dieser drei Kriterien, Algorithmen und Komplexitätsangaben für das "Query Containment" Problem vorgestellt.
Abstract

The query containment problem is to check if the answer set of one query is always a subset of another query for any given database. Algorithms for query containment are crucial in several contexts in databases.

As far as the query containment problem is concerned, various algorithms and complexity results can be obtained, depending on the following three factors:

- the query language,
- whether some constraints are considered,
- the structure of the database.

In this work, we address the query containment problem from the above three different aspects by giving algorithms and complexity results.
Contents

I Introduction ................................................. 1

1 Introduction ............................................ 3
  1.1 Motivation ........................................ 3
  1.2 Three Factors Deciding Query Containment .......... 4
    1.2.1 Factor 1: The query language ................. 4
    1.2.2 Factor 2: Whether some constraints are considered 4
    1.2.3 Factor 3: The structure of the database ........ 5
  1.3 Detailed Outline .................................. 7
    1.3.1 Containment of Queries on Relational Databases 8
    1.3.2 Containment of Queries on Tree Databases .... 9

II Query Containment on Relational Databases .............. 13

2 Preliminaries ............................................. 15
  2.1 Relational Queries ................................ 15
    2.1.1 Conjunctive queries (CQs) .................... 15
    2.1.2 Unions of CQs ................................ 16
    2.1.3 CQs with Inequalities ......................... 17
    2.1.4 CQs with Negated Subgoals .................... 18
    2.1.5 Acyclic CQs ................................ 18
    2.1.6 Datalog ..................................... 19
  2.2 Containment Checking of CQs under Integrity Constraints 21

3 Containment of Conjunctive Queries with Safe Negation .... 23
  3.1 Introduction .................................... 23
  3.2 Preliminaries .................................... 26
    3.2.1 Query Containment ............................ 26
    3.2.2 The Containment Checking Algorithm for CQs .... 26
  3.3 Query Containment for $CQ^\sim$s .................. 27
    3.3.1 Some Necessary Conditions ................... 27
A.1.1 Kripke Structures ........................................ 131
A.1.2 PLTL ...................................................... 132
A.1.3 CTL and CTL* ............................................ 133
A.2 Tree Model Property ....................................... 134
   A.2.1 Decidability of SnS .................................... 135
   A.2.2 Bisimulation ............................................. 136
   A.2.3 Tree Model Property of Temporal Logics .......... 138
   A.2.4 Guarded Logics and Acyclic CQs .................... 138

B Monadic Datalog and Regular Expressions 141

C Lists ......................................................... 147
   List of Theorems ............................................. 147
   List of Definitions ......................................... 149
   List of Examples ............................................ 150
   List of Algorithms ......................................... 151
   List of Figures ............................................... 151
   List of Tables ............................................... 152

Bibliography .................................................. 152
Part I

Introduction
1 INTRODUCTION

1.1 Motivation

The query containment problem checks, for any given database, if the answer set of one query is always a subset of another query. Algorithms for query containment are crucial in several contexts in databases.

In a relational database system, given a database $D$ and a query $Q$, the query complexity (the size of $D$ is considered as the input variable) of the query evaluation $Q(D)$ is NP-complete. The join operation is the main reason for the intractability. As a result, one task for query optimization in a database system is to minimize the join size of the query while keeping the optimized query semantically equivalent to the old one.

The main motivation of query containment also lies in its tight relation to the problem of answering queries using views [LMSS95, AD98], which arises as a central problem in data integration and data warehousing. Furthermore, query containment has also been used for checking integrity constraints [GSUW94] and for deciding query independence of updates [LS93].

**Definition 1 (Query equivalence)** Given two queries $Q_1$ and $Q_2$, $Q_1$ is equivalent to $Q_2$, written as $Q_1 \equiv Q_2$, if and only if for any database $D$, $Q_1(D) \equiv Q_2(D)$.

**Definition 2 (Query containment)** Given two queries $Q_1$ and $Q_2$, $Q_1$ is contained in $Q_2$, written as $Q_1 \subseteq Q_2$, if and only if for any database $D$, $Q_1(D) \subseteq Q_2(D)$.

**Definition 3 (Containment and equivalence)** $Q_1 \equiv Q_2$ if and only if $Q_1 \subseteq Q_2$ and $Q_2 \subseteq Q_1$.

As far as the query containment problem is concerned, various algorithms and complexity results can be obtained depending on the three factors described in the following section.
1.2 Three Factors Deciding Query Containment

1.2.1 Factor 1: The query language

Conjunctive queries (CQs) are considered by Chandra and Merlin [CM77], who showed that the query containment of CQs is NP-complete. This NP-complete result holds also for the query evaluation problem of CQs. This equivalence is justified by the fact that for positive CQs, one has to only test one canonical database to know whether one query is contained in another. This is also called the homomorphism property.

However, the homomorphism property does not hold anymore if the CQs are extended with built-in predicates (CQ\#s), such as "\(<\)" and "\(\neq\)", or if negated subgoals (CQ\~s) are allowed in CQs. In order to check whether \(Q_1 \subseteq Q_2\), one has to test an exponential number of canonical databases of \(Q_1\). As a result, the complexity of containment checking for CQs with extensions is higher than NP. In fact, the containment problem of both \(CQ\#\)s and \(CQ\~\)s are \(\Pi^p_2\)-complete.

If the CQs are further extended with aggregation functions, such as average, maximum or count, which are generally supported in RDBMS, the containment checking has higher complexity [CNS99].

If the expressiveness of the query language is beyond first-order logic (a.k.a. relational algebra, relational calculus), the containment checking problem is undecidable. The undecidability of containment for Datalog was proven by Shmueli [Shm87] with the reduction from the equivalence problem of context-free grammars.

Given any conjunctive query, a hypergraph can be obtained from it, illustrating the join relations among subgoals. If the hypergraph is acyclic, the CQ is called acyclic. Query evaluation of acyclic CQs is in polynomial time, and it is also the case for query containment of acyclic CQs. However, there exists little research on the containment checking of acyclic CQs with extensions. In this thesis a Co-NP-complete result is given for containment of acyclic CQs with safe negation.

1.2.2 Factor 2: Whether some constraints are considered

The integrity constraints in database systems range from the well-known functional dependencies to the complicated ones represented with various logic formalisms. Query containment algorithms need to be modified if the integrity constraints are included in the database schema. This kind of containment checking is normally denoted as \(\subseteq_{ic}\), where \(ic\) means integrity constraints. If the query containment under such integrity constraints is considered, all databases consistent with the
1.2. THREE FACTORS DECIDING QUERY CONTAINMENT

integrity constraints have to be checked.

The query containment problem in the presence of integrity constraints is studied first especially with functional and inclusion dependencies [JK82]. Later, Zhang et al. [ZÖ97] extended the integrity constraints to implication constraints and referential constraints, which are the generalized form of functional and inclusion dependencies respectively. Generally, some integrity constraints can be expressed in the form of CQs as follows.

\[ ic: \text{panic} :- \text{emp}(E, D, S), S > 10000. \]

where the the conjunction of subgoals represents the violation of the constraints. In the above example, the constraint says that every employee should have less than or equal to the salary of 10000$/\text{month}$. The technique of containment checking under such constraints is as follows: to test whether \( Q_1 \subseteq_{ic} Q_2 \), one has to check whether \( Q_1 \) is contained in \( Q_2 \cup ic \). If the containment checking \( Q_1 \subseteq_{ic} Q_2 \) is successful, then containment \( Q_1 \subseteq_{ic} Q_2 \) holds too because \( Q_1 \) is not consistent with the integrity constraints. Consequently, an empty answer set would be returned if \( Q_1 \) is applied to any database \( D \) which is consistent with \( ic \).

However, there are many other constraints that could not be expressed with the above form.

In this thesis, we propose an algorithm for conjunctive query containment in the presence of disjunctive integrity constraints.

1.2.3 Factor 3: The structure of the database

Finite databases (logically speaking, structures) can be ordered or unordered. Over ordered databases, query languages usually have more expressiveness than on unordered ones. For instance, over ordered structures, the fixpoint logic (FO(\text{FP})) queries express exactly \text{PTIME}. However, if the structures are not ordered, the same logic can not even express simple queries like "the parity of a set" [Via01].

More interestingly, some properties such as transitive closure can be expressed with first-order logic if the underlying database is ordered. It is well-known that regular expressions over words can express transitive closure properties like \( a^* \) or \( a^+ \), while the equivalence problem of regular expressions is \text{Co-NP}-complete.

By introducing Datalog to indicate that some properties are not expressible in first-order logic, the following two example databases are always deployed: \textit{flights} (Fig. 1.1) and \textit{family trees} (Fig. 1.2).

As usual, the transitive closure is asked for both databases. For instance,
1. INTRODUCTION

- $Q_1$: from the flight database, returns every city which is reachable from London, and
- $Q_2$: from the family tree database, returns every descendant of queen Victoria.

The difference between these two databases is that one is a graph, while the other is an ordered tree. It is proven that $Q_1$ is not expressible in first-order logic [AHV95]. However, for the family tree database, an order can be given for each node with a starting tag and an end tag (cf. Chapter 6.1.1), so that the descendant query can be easily expressed in conjunctive queries with built-in predicates. Consequently, the results can be obtained with any standard SQL system.

Query equivalence, similar to query evaluation, benefits from the order too. For instance, the containment test of Datalog programs is undecidable, and this undecidability is so robust that there are only a few fragments of Datalog whose containment test is proven decidable (e.g. Monadic datalog). Since over ordered structures many seemingly recursive features turn out to be first-order expressible, the corresponding query containment is also decidable and some are even tractable.

![Diagram of a flight network showing flights between London, New York, Cairo, and Tokyo.]

Figure 1.1: The flight example

Internet databases such as XML and LDAP are generally represented as ordered labeled trees. The corresponding query model XPath can be proven to be first-order expressible in spite of many recursive features like the ancestor-descendant relation. As a result, the query containment problem of XPath queries on XML can be reduced to that of CQs with linear constraints.

However, the containment is different from the general containment because containment checking on such tree databases considers every database which is consistent with the tree model property. We call this containment tree containment, denoted as $\subseteq_{\text{tree}}$. Figure 1.3 illustrates that three XPath expressions are generally not equivalent, if arbitrary underlying structures are considered. However, they are tree equivalent, if only tree structures are considered. As a result, a special
algorithm has to be designed for the tree containment.

In this thesis, we consider the query containment problem over ordered structures, such as finite words and finite ordered trees.

\[
\begin{array}{ccc}
  & a & a \\
  &   & a \\
  b & | & b \\
  \_ & | & \_ \\
  \_ & | & \_ \\
  \_ & | & \_ \\
\end{array}
\]

(a) (b) (c)

Figure 1.3: Queries (a), (b) and (c) are tree equivalent.

### 1.3 Detailed Outline

In this work, we address the query containment problem from the above three different aspects by giving algorithms and complexity results.
1.3.1 Containment of Queries on Relational Databases

In Chapter 2, we recall the necessary definitions concerning query containment of conjunctive queries (CQs), CQs with built-in predicates, CQs with safe negations, and acyclic CQs respectively. Furthermore, we give a brief introduction on containment of CQs under various integrity constraints. Note that since query containment is just the sub-problem of query equivalence, we use them interchangeably in the later parts of the thesis.

In Chapter 3 we solve the problem of containment checking of conjunctive queries with safe negation.

The containment problem for conjunctive queries with safe negated subgoals has drawn considerably less attention in the past. Levy and Sagiv discussed uniform containment [LS93], which is a sufficient, however not necessary condition for containment. Ullman [Ull97] argued that the complexity of the containment test is $\Pi_2$-complete, and also proposed an algorithm based on the approach of canonical databases. The drawback of the algorithm is that if the containment holds, one has always to check an exponential number of canonical databases.

We propose a new method to solve the general query containment problem for conjunctive queries with safe negated subgoals ($CQ^-$s). Given two $CQ^-$s $Q_1$ and $Q_2$, and their positive counterparts $Q_1^+$ and $Q_2^+$ (definitions in Section 3.3.1), we show that there are two factors deciding the complexity of the problem $Q_1 \subseteq Q_2$:

- $Q_1^+ \sqsubseteq Q_2^+$? This is a necessary condition.
- the number of containment mappings from $Q_2^+$ to $Q_1^+$.

In comparison to the existing algorithm, which requires always an exponential number of canonical database to be tested to prove the result $Q_1 \subseteq Q_2$, the algorithm proposed in this chapter exploits the containment mappings from $Q_2^+$ to $Q_1^+$, and it terminates when the specified tests succeed. We show that in the worst case the algorithm has the same performance as the existing algorithm. Our algorithm also extends naturally to unions of $CQ^-$s.

In Chapter 4 we continue searching for the solution of the containment checking problem for acyclic $CQ^-$s. As mentioned before, if the CQs are acyclic, the query evaluation (query complexity) is reduced from NP-complete to polynomial time. This polynomial time complexity holds also for containment checking of CQs. As far as the extensions of CQs are concerned, the query evaluation stays in polynomial time, which is unfortunately not the case for containment checking.

Consider the acyclic CQs with safe negation, it is not difficult to get an upper bound of Co-NP, since the general algorithm of containment checking always follows
1.3. **DETAILED OUTLINE**

the theorem:

**Theorem 1 (Conjunctive Query Containment)** Given two queries \( Q_1 \) and \( Q_2 \),
\( Q_1 \subseteq Q_2 \) if and only if for every canonical database \( D \) of \( Q_1 \), the answer set of
\( Q_2(D) \) is subset of \( Q_1(D) \).

For containment of (positive) CQs, one canonical database has to be checked.
However, if the extensions are considered, an exponential number of canonical
databases need to be checked. Since the query evaluation is reduced each time
from NP-complete to PTIME for acyclic queries, the complexity is easily proven
to be in Co-NP.

To prove the lower bound, we make a reduction from the co-problem of 3SAT,
which is Co-NP complete, to the containment problem of acyclic CQ’s. The proof
shows that under the universe \( \{0, 1\} \), the containment checking is Co-NP hard.
The hardness proof for the containment checking problem under any universe is
left open.

In Chapter 5 we propose a containment algorithm for CQs with incomplete
information, which is represented with disjunctive integrity constraints.

For instance, the following disjunctive constraint states that each employee
belongs to one of the two groups, \( p_1 \) and \( p_2 \).  
\[
\text{drc: } \text{empl}(X) \rightarrow \exists W_1 \text{ member}(X, p_1, W_1) \vee \exists W_2 \text{ member}(X, p_2, W_2). 
\]

Given two CQs \( Q_1 \) and \( Q_2 \), to test whether \( Q_1 \subseteq Q_2 \) under such disjunctive
integrity constraints, one has to first expand the query \( Q_1 \) with the disjunctive
rules to a set of sub-queries, then test whether all the expanded sub-queries are
contained in \( Q_2 \).

We show that if the disjunctive rules are acyclic - a property which can be
checked efficiently - then the expansion will always terminate.

### 1.3.2 Containment of Queries on Tree Databases

In the rest of the thesis we consider the containment problem with queries over
ordered databases. With the popularity of Internet applications, the exchange
data model XML has recently attracted intensive research. In comparison to the
relational data model, which is normally unordered, XML is in essence an ordered,
labeled and unranked tree. XPath, as the most deployed path query model on
XML, is surprisingly efficient [GK02b]. In fact, it has a linear time query evaluation
complexity. Naturally, one is interested in the containment problem of XPath
queries. However, we encounter the following difficulties:
1. INTRODUCTION

- XPath queries involve recursion, which is normally not expressible in first-order logic,
- the query automata, or MSO on trees, are obviously too strong to represent XPath,
- existing results on fragments of XPath query containment differ significantly on the formalism basics. For instance, the Co-NP-complete result of containment on XPath fragment XP(///,*]) is proven by using both tree pattern matching and tree automata. Other results on containment of various XPath extensions deploy regular tree grammars [Woo03], tree automata [NS03], chase algorithms [DT01], and regular path expressions [CGV03] respectively. None of the previous work gave an exact expressive power of XPath ¹, and a uniform theoretical formalism is missing.

Dealing with the query languages over ordered trees, there are several new questions (except for the acyclicity property) that have been hardly encountered when only the unordered relational databases are considered. One of these is for instance, how to encode the ordered structures. Finite words can be considered as a special fragment of finite ordered trees that do not have branches. The knowledge of query languages applied to finite words is crucial for the further understanding of ordered trees. As a result, we consider queries over finite words in Chapter 7.

In Chapter 7, we first define regular languages and star-free regular languages, which is followed by the mechanism for encoding the finite words as linearly ordered databases.

We then concentrate on the fragment of XPath over finite words: XP(/,//,*), which contains child and descendant axes, as well as wildcards. We show that XP(/,//,*]) is a fragment of star-free regular languages, which implicates that it belongs to a fragment of FO. As far as the containment test of XP(/,//,*]) is concerned, the results of equivalence problem for star-free regular languages can not be directly applied, since the complexity of the latter is Co-NP complete. We propose an algorithm by first rewriting the queries into normal form and then applying the containment mappings. Due to the acyclicity and homomorphism properties of the queries, the algorithm is in polynomial time.

In Chapter 8, we consider the containment problem of XPath with all axes. We show that XPath queries form a fragment of Conjunctive Queries with Linear Constraints on Trees (CQLC) and address the containment checking problem on such queries. In the context of containment checking of queries we show that the homomorphism property still holds on queries with axes child, descendant and

¹Recently, Marx [Mar04] solved the expressive power problem.
1.3. **DETAILED OUTLINE**

following. As a result, the containment checking of this fragment is tractable.

The containment problem that we consider in this work is however restricted to *tree databases*. In order to find a method for *tree containment checking*, we propose a *wildcards elimination* algorithm, which first calculates the closure of the constraints by a set of axioms and then eliminates the related information of wildcard nodes. We prove that the query after the wildcards elimination processing is nevertheless *tree equivalent* to the original query and thus the general containment checking algorithm can be applied.

We present some concluding remarks in **Chapter 9**.

Some of the results presented in this work have already been published in several international conferences and workshops. Main results in Chapter 3 are published in [WL03] and [LW03]. The results in Chapter 5 have been published in [WL02].
Part II

Query Containment on Relational Databases
2 PRELIMINARIES

2.1 Relational Queries

2.1.1 Conjunctive queries (CQs)

Conjunctive queries, a.k.a. select-project-join (SPJ) queries, a.k.a. positive existential first-order logic, are the backbone of relational query languages, such as SQL, and enjoy several desirable properties, such as decidability (which is not the case for first-order logic). A conjunctive query (CQ for short) is denoted by a rule:

\[ h(\bar{X}) : \leftarrow p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n). \]

where \( h, p_1, \ldots, p_n \) are predicates whose arguments are variables or constants, \( h(\bar{X}) \) is the head, \( p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n) \) are the positive subgoals. Each subgoal \( p_i(\bar{X}_i) \) includes a relation \( p_i \) and a tuple of arguments \( \bar{X}_i \) corresponding to the relation schema. The variables \( \bar{X} \) are called distinguished variables. The query is safe if every distinguished variable appears in the body.

A CQ is applied to a set of finite database relations by considering all possible substitutions of values for the variables in the body. If a substitution makes all the positive subgoals true, then the same substitution, applied to the head, composes one answer of the conjunctive query. The set of all answers to a query \( Q \) with respect to a certain database \( D \) is denoted by \( Q(D) \).

Definition 4 (Query containment and equivalence) a CQ \( Q_1 \) is contained in another one \( Q_2 \), denoted as \( Q_1 \subseteq Q_2 \), if for all databases \( D \), \( Q_1(D) \subseteq Q_2(D) \). Two CQs are equivalent if and only if they are contained in each other. \( \square \)

An algorithm for checking the containment of CQs was proposed by Chandra and Merlin [CM77].

Lemma 1 (Containment of CQs [CM77]) Consider two CQs \( Q_1 \) and \( Q_2 \):

\[ Q_1 : h(\bar{X}) : \leftarrow p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n). \]
\[ Q_2 : h(\bar{U}) : \leftarrow q_1(\bar{U}_1), \ldots, q_l(\bar{U}_l). \]
Then $Q_1 \subseteq Q_2$ if and only if there exists a containment mapping $\rho$ from the variables of subgoals in $Q_2$ to those in $Q_1$, such that $\{\rho(q_1(\bar{U}_1)), \ldots, \rho(q_n(\bar{U}_n))\} \subseteq \{p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\}$, and $\rho(h(\bar{U})) = h(\bar{X})$. \hfill \square

When the heads of both $Q_1$ and $Q_2$ do not contain variables, they are called boolean queries. It is obvious that boolean queries are the generalized forms of normal queries. The containment problem for CQs is shown to be NP-complete [CM77].

**Example 1 (Containment mapping)** Consider the queries $Q_1$ and $Q_2$:

$$
Q_1 : \quad q(X, Z) \leadsto a(X, Y), a(Y, Z). \\
Q_2 : \quad q(A, C) \leadsto a(A, B), a(B, C), a(B, D).
$$

There is one and only one containment mapping from $Q_2$ to $Q_1$:

$$
\{A \rightarrow X, B \rightarrow Y, C \rightarrow Z, D \rightarrow Z\}
$$

As a result, $Q_1 \subseteq Q_2$ holds.

### 2.1.2 Unions of CQs

Unions of conjunctive queries have the following defined form.

**Definition 5 (Union of CQs [Ull89])** Let $Q = Q_1 \cup \ldots \cup Q_n$ be a union of CQs, in which $Q_1, \ldots, Q_n$ have a common head predicate.

- Given any database $D$, $Q(D) = Q_1(D) \cup \ldots \cup Q_n(D)$.
- Let $Q$ be a CQ and $Q$ be a union of CQs. $Q \subseteq Q$ if $Q(D) \subseteq Q(D)$ for any given database $D$.

If only conjunctive queries are allowed (without any extensions, which we introduce in the next sections), the query containment algorithm is similar to the one from Chandra and Merlin.

**Theorem 2 (Containment of unions of CQs [SY80])** Let $Q$ be a CQ and $Q = Q_1 \cup \ldots \cup Q_n$ be a union of CQs, then $Q \subseteq Q$ if and only if there is a $Q_i (1 \leq i \leq n)$ such that $Q \subseteq Q_i$. \hfill \square

However, this theorem does not hold anymore if some extensions are included in the CQs.
2.1. RELATIONAL QUERIES

2.1.3 CQs with Inequalities

If inequalities (or built-in predicates) are allowed in a conjunctive query, the form of CQ is extended as follows:

\[ q(\bar{X}) \leftarrow p_1(\bar{Y}_1), \ldots, p_n(\bar{Y}_n), I. \]

where \( I \) is the conjunct of formulas of the form \((u_1 \text{ op } u_2)\), in which both \( u_1 \) and \( u_2 \) can be constants or variables, and if any \( u \) is a variable, then \( u \) is in \( \{\bar{Y}_1, \ldots, \bar{Y}_n\} \).

The containment definition of CQs with inequalities is the same as that of CQs. Theorem 3 states the algorithm of containment checking.

**Theorem 3 (Containment of CQs with Inequalities [Z{"O}97])** Consider two CQs \( Q_1 \) and \( Q_2 \):

\[
Q_1 : h(\bar{X}) \leftarrow p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n), I_1. \\
Q_2 : h(\bar{U}) \leftarrow q_1(\bar{U}_1), \ldots, q_l(\bar{U}_l), I_2.
\]

Let \( \rho_1, \ldots, \rho_m \) be all the containment mappings from the variables of subgoals in \( Q_2 \) to those in \( Q_1 \), such that \( \{\rho(q_1(\bar{U}_1)), \ldots, \rho(q_l(\bar{U}_l))\} \subseteq \{p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\} \), and \( \rho(h(\bar{U})) = h(\bar{X}) \). Then \( Q_1 \subseteq Q_2 \) if and only if the following holds:

\[
I_1 \implies \rho_1(I_2) \vee \ldots \vee \rho_m(I_2)
\]

\[
\square
\]

**Example 2 (Containment mapping)** Consider the queries \( Q_1 \) and \( Q_2 \):

\[
Q_1 : \text{q()} \leftarrow \text{a(A, B), a(B, A)}. \\
Q_2 : \text{q()} \leftarrow \text{a(X, Y), X \leq Y}.
\]

There are two containment mappings from \( Q_2 \) to \( Q_1 \):

\[
\rho_1 : \{X \rightarrow A, Y \rightarrow B\} \\
\rho_2 : \{X \rightarrow B, Y \rightarrow A\}
\]

and

\[
\text{true} \rightarrow \rho_1(X \leq Y) \vee \rho_2(X \leq Y) \\
\text{true} \rightarrow (A \leq B) \vee (B \leq A)
\]

\[
\square
\]

As a result, \( Q_1 \subseteq Q_2 \) holds.

The complexity of containment for CQs with inequalities is \( \Pi_2^p \) complete. The upper and lower bounds are given by Klug [Klu88] and van der Meyden [vdM92] respectively.
2.1.4 CQs with Negated Subgoals

A *conjunctive query with negation* (CQ⁻) extends a *conjunctive query* CQ by allowing negated subgoals in the body. It has the following form:

\[ h(\bar{X}) \leftarrow p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n), \neg s_1(\bar{Y}_1), \ldots, \neg s_m(\bar{Y}_m). \]

where \( h, p_1, \ldots, p_n, s_1, \ldots, s_m \) are predicates whose arguments are variables or constants, \( h(\bar{X}) \) is the head, \( p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n) \) are the positive subgoals, and \( s_1(\bar{Y}_1), \ldots, s_m(\bar{Y}_m) \) are the negated subgoals. We assume that, firstly, the variables occurring in the head also occur in the body; secondly, all the variables occurring in the negated subgoals also occur in positive ones, which is also called the *safeness* condition for \( CQ^- \). The examples in this thesis are *safe CQ⁻*s if not mentioned otherwise.

A \( CQ^- \) is applied to a set of finite database relations by considering all possible substitutions of values for the variables in the body. If a substitution makes all the positive subgoals true and all the negated subgoals false (i.e., they do not exist in the database), then the same substitution, applied to the head, composes one answer of the conjunctive query. The set of all answers to a query \( Q \) with respect to a certain database \( D \) is denoted by \( Q(D) \).

The complexity of containment for CQs with safe negation is \( \Pi^p_2 \)-complete [Ull97].

2.1.5 Acyclic CQs

Given a conjunctive query \( Q \), the hypergraph \( H(Q) \) to it is defined as \( H(Q) = (V, E) \), where the set \( V \) of vertices consists of all variables occurring in the body of \( Q \), while the set \( E \) of hypergraph contains, for each atom \( A \) in the rule body, the set \( \text{var}(A) \) of all variables occurring in \( A \).

A query \( Q \) is acyclic if its associated hypergraph \( H(Q) \) is acyclic. The join tree \( T(Q) \) derived from an acyclic CQ \( Q \) is an undirected tree whose vertices are the atoms in the body of \( Q \) such that [AHV95]:

1. Each edge \( A_1 \) and \( A_2 \) is labeled by the set of variables \( \text{var}(A_1) \cap \text{var}(A_2) \); and
2. For every pair \( A_1, A_2 \) of distinct nodes, for each \( X \in \text{var}(A_1) \cap \text{var}(A_2) \), each edge along the unique path between \( A_1 \) and \( A_2 \) includes label \( X \).

Example 3 (Acyclic CQ) Consider the following query \( Q \),

\[ Q : q() \leftarrow S_1(A, B, C), S_2(C, D, E), S_3(A, F, E), S_4(A, C, E). \]

the hypergraph and join tree are depicted in Figure 2.1.
2.1. RELATIONAL QUERIES

Figure 2.1: The hypergraph and join tree of Example 3

Note that the root of the join tree of an acyclic \( CQ \) is not necessarily unique. For example, if the hypergraph of a \( CQ \) is a chain (cf. Example 4), then there exist two trees with different roots.

**Example 4 (Conjunctive query with the chain form)** Consider the following query \( Q_1 \),

\[
Q_1 : q() \quad \leftarrow \quad r_1(A, B), r_2(B, C), r_3(C, D).
\]

Since the join tree of the query is a chain, the root can be either \( r_1(A, B) \), or \( r_3(C, D) \).

**Proposition 1 (Acyclicity and join tree [AHV95])** A conjunctive query \( Q \) is acyclic iff it has a join tree.

2.1.6 Datalog

A datalog query is a Horn-clause program without function symbols. A datalog query on a database \( D \) is composed of a set of rules, and each rule has the form:

\[
h(\bar{X}) \quad \leftarrow \quad p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n).
\]

Each \( p_i \) in the body is a predicate on a relation. A predicate whose relation is stored in the database \( D \) is called an extensional database (EDB) relation, while one defined by the rules (it appears on the head of a rule) is called an intensional database (IDB) relation. A particular IDB predicate is designated as a goal predicate, and the answers to the query are all facts about the goal predicate that can be deduced from the EDB relations. For instance, suppose we have a undirected graph database with one relation \( E \). A tuple \( \text{edge}(X, Y) \) means that there is an edge from vertex \( X \) to \( Y \). The following is a datalog query:
Example 5 (Datalog with transitive closure)

\[
\begin{align*}
  r_1 & : \text{trans}(X, Y) \leftarrow \text{edge}(X, Y). \\
  r_2 & : \text{trans}(X, Y) \leftarrow \text{edge}(X, Z), \text{trans}(Z, Y).
\end{align*}
\]

The query has two rules, \( r_1 \) and \( r_2 \). It has an EDB predicate \( \text{edge} \), and an IDB predicate \( \text{trans} \). Each tuple \((x_0, y_0)\) in \( \text{trans} \) means that vertex \( y_0 \) is connected with vertex \( x_0 \). As shown by the query above, a datalog query can be recursive. A conjunctive query can be viewed as a single-rule datalog query.

It is obvious that Datalog is more expressive than CQs or first-order logic. For instance, the transitive closure expressed in Example 5 is not expressible in SQL. This language is essentially a fragment of fixpoint logic interpreted over finite structures, i.e., databases.

The gain of expressive power does not, however, come for free. Query evaluation of Datalog programs is higher than that of CQs [Var82]. To be accurate, the data complexity is LOGSPACE-complete for CQs and P-complete for Datalog while the combined complexity is NP-complete for CQs and EXPTIME-complete for Datalog. Shmueli [Shm87] proved that the containment problem of Datalog programs is undecidable, by a reduction from the equivalence problem of context-free languages.

Monadic datalog

A datalog program is a monadic program, if its IDB predicates are monadic, and the EDB predicates may have arbitrary arities. For instance, the following Datalog program is monadic Datalog.

Example 6 (Monadic Datalog)

\[
\begin{align*}
  r_1 & : \text{access}(X) \leftarrow \text{source}(X). \\
  r_2 & : \text{access}(X) \leftarrow \text{access}(Y), \text{access}(Z), \text{triple}(X, Y, Z).
\end{align*}
\]

The containment checking problem of monadic datalog is decidable [CGKV88]. It is shown that it is in doubly exponential time and EXPTIME hard. This is the only decidable result on containment checking of fragments of datalog programs, in which both programs are recursive. Recently, Gottlob and Koch [GK02a] proved that Monadic datalog on trees has the same expressive power as MSO on trees. We will elaborate this result in Chapter 7.
2.2. CONTAINMENT CHECKING OF CQS UNDER INTEGRITY CONSTRAINTS

Linear datalog

A datalog program is linear, if in each rule of the program, there can be at most one relation that is mutually recursive with the head relation. The program in Example 5 is linear, while the one in Example 6 is not, because the IDB access appears two times in rule $r_2$. The containment checking problem of linear datalog programs is unfortunately undecidable, even for very simple programs with only binary predicates.

2.2 Containment Checking of CQs under Integrity Constraints

The query containment problem in the presence of integrity constraints is studied first in [JK82], especially with functional and inclusion dependencies. The chase algorithm, which was applied in the reasoning of integrity constraints of relational schema, can be deployed by the containment checking.

The general idea is as follows: given queries $Q_1$ and $Q_2$, and the integrity constraints, such as function dependencies $F$. To test whether $Q_1 \sqsubseteq_F Q_2$, one has to first expand $Q_1$ with the dependency rules from $F$, and a new query $Q'_1$ is obtained after the chase process. The normal containment checking of $Q_2 \sqsubseteq Q'_1$ can then be applied.

However, the chase method can not be generalize to all integrity constraints. In [ZÖ97], the integrity constraints are extended to implication constraints and referential constraints, which are the generalized form of functional constraints and inclusion dependencies, respectively.
2. PRELIMINARIES
3 CONTAINMENT OF CONJUNCTIVE QUERIES WITH SAFE NEGATION

3.1 Introduction

This chapter considers the problem of query containment of conjunctive queries (CQs) with safe negated subgoals CQ¯s. The query containment problem is to check whether the answer set of one query is always a subset of another query for all databases. Algorithms for query containment are of interest in several contexts in the database area.

Recently, there is a renewed interest in containment checking of conjunctive queries. The main motivation lies in its tight relation to the problem of answering queries using views [LMS95, AD98], which arises as the central problem in data integration and data warehousing (see [Ull97] for a survey). Furthermore, query containment has also been used for checking integrity constraints [GSUW94], and for deciding query independence of updates [LS93].

Based on the NP-completeness result proposed by Chandra and Merlin [CM77], many researchers have been working on extensions of the containment question. Containment of CQs with inequalities is discussed in [Klu88, ZO93]. Containment of unions of CQs is treated in [SY86], containment of CQs with negated subgoals in [LS93, Ull97], containment over complex objects in [LS97], and over semi-structured data with regular expressions in [FLS98].

The containment problem for conjunctive queries with safe negated subgoals has drawn considerably less attention in the past. In [LS93] uniform containment is discussed, which is a sufficient, however not necessary condition for containment. In [Ull97] it is argued that the complexity of the containment test is \( \Pi_2^p \)-complete. An algorithm based on the approach of canonical databases was sketched which tests an exponential number of canonical databases. The following example explains the approach:

Example 7 (Containment of CQ¯s) Consider the following queries \( Q_1 \) and

23
Table 3.1: The five canonical databases and their answers to $Q_1$ and $Q_2$

<table>
<thead>
<tr>
<th>Partition</th>
<th>Canonical Databases</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>${X}{Y}{Z}$</td>
<td>${a(0,1), a(1,2)}$</td>
<td>$Q_1: q(0,2); Q_2: q(0,2)$</td>
</tr>
<tr>
<td>${X, Y}{Z}$</td>
<td>${a(0,0), a(0,1)}$</td>
<td>$Q_1: false$</td>
</tr>
<tr>
<td>${X}{Y, Z}$</td>
<td>${a(0,1), a(1,1)}$</td>
<td>$Q_1: false$</td>
</tr>
<tr>
<td>${X, Z}{Y}$</td>
<td>${a(0,1), a(1,0)}$</td>
<td>$Q_1: q(0,0); Q_2: q(0,0)$</td>
</tr>
<tr>
<td>${X, Y, Z}$</td>
<td>${a(0,0)}$</td>
<td>$Q_1: false$</td>
</tr>
</tbody>
</table>

$Q_2$: 

$Q_1: q(X, Z) \Rightarrow a(X, Y), a(Y, Z), \neg a(X, Z).$

$Q_2: q(A, C) \Rightarrow a(A, B), a(B, C), a(B, D), \neg a(A, D).$

In order to show that $Q_1 \subseteq Q_2$, the approach in [Ull97] considers all five partitions of $\{X, Y, Z\}$ in Table 3.1: all variables in one set of a certain partition are replaced by the same constant. From each partition, a canonical database, built out of the positive subgoals, is generated according to the predicates in the body of $Q_1$. At first, $Q_1$ has to be applied to the canonical database $D$ from each partition, and if the answer set is not empty, then the same answer set has to be obtained from $Q_2(D)$. Next, for each canonical database $D$ which results in a nonempty answer, we have to extend it with “other tuples that are formed from the same symbols as those in $D$”. In fact, for each specific predicate, let $k$ be the number of arguments of the predicate, $n$ be the number of symbols in the canonical database, and $r$ be the number of subgoals of $Q_1$ (both positive and negative), there will be $2^{(n-k-r)}$ sets of tuples which have to be checked. Taking the partition $\{X\}\{Y\}\{Z\}$, we need to consider 6 other tuples: $\{a(0,0), a(1,0), a(1,1), a(2,0), a(2,1), a(2,2)\}$. At the end, one has to check $2^6$ canonical databases, and if for each database $D'$, $Q_2(D')$ yields the same answer as $Q_1$, it can then be concluded that $Q_1 \subseteq Q_2$, which is true in this example.

Example 8 (Containment of $CQ^-$s) Consider the following queries $Q_1$ and $Q_2$:

$Q_1: q(X, Z) \Rightarrow a(X, Y), a(Y, Z), \neg a(X, Z).$

$Q_2: q(A, C) \Rightarrow a(A, B), a(B, C), \neg b(C, C).$

This example differs from Example 7 by the negated subgoal. The application of $Q_2$ to the canonical databases shown in Table 3.1 yields the same answer as $Q_1$. 
3.1. INTRODUCTION

Similar to the above example, extra tuples have to be added into the canonical database. Taking the partition \{X\}\{Y\}\{Z\}, we have 15 other tuples (9 tuples with \(b(0,0), \ldots, b(2,2)\) and 6 tuples as in Example 7), such that \(2^{15}\) canonical databases have to be verified. Since the database \(D = \{a(0,1), a(1,2), b(2,2)\}\) is a counter-example such that \(Q_2(D)\) does not generate the same answer as \(Q_1(D)\), the test terminates with the result \(Q_1 \not\subseteq Q_2\).

In this chapter, we propose a new method to solve the general query containment problem for conjunctive queries with safe negated subgoals (CQ’s). Given two CQ’s \(Q_1\) and \(Q_2\), and their positive counterparts \(Q_1^+\) and \(Q_2^+\) (definitions in Section 3.3.1), we show that there are two factors deciding the complexity of the problem \(Q_1 \subseteq Q_2\):

- \(Q_1^+ \subseteq Q_2^+\)? This is a necessary condition.
- the number of containment mappings from \(Q_2^+\) to \(Q_1^+\).

Comparing to the algorithm described in [Ull97], which requires always an exponential number of canonical database to be tested to prove the result \(Q_1 \subseteq Q_2\), the algorithm proposed in this chapter exploits the containment mappings from \(Q_2^+\) to \(Q_1^+\), and terminates when the specified tests succeed. We show that in the worst case, the algorithm has the same performance as the one proposed in [Ull97]. Our algorithm also extends naturally to unions of CQ’s. Due to the close relation between query containment and answering queries using views, we give some notes on considering answering queries using views when both queries and views have safe negated subgoals allowed.

The rest of the chapter is organized as follows: in Section 3.2 we recall the definition of a CQ’s and containment of both CQ’s and CQ’s. In Section 3.3, we first prove two necessary conditions for the containment test of CQ’s, which is then followed by the main theorem of the chapter. The proof of correctness and completeness is given as well. In Section 3.4 the theorem and the algorithm based on it are naturally extended to the containment test of unions of CQ’s. In Section 3.5 we discuss the issues of answering queries using views when both queries and views have negated subgoals allowed. Finally the conclusions and future work are presented.
3.2 Preliminaries

3.2.1 Query Containment

Unlike CQs with only positive subgoals, which are always satisfiable, CQ$^-$s might be unsatisfiable.

**Proposition 2 (Satisfiability of CQ$^-$)** A CQ$^-$ is unsatisfiable if and only if there exist $p_i(\tilde{X}_i)(1 \leq i \leq n)$ and $s_j(\tilde{Y}_j)(1 \leq j \leq m)$ such that $p_i = s_j$ and $\tilde{X}_i = \tilde{Y}_j$.

From now on, we only refer to satisfiable CQ$^-$s, if not otherwise mentioned.

The containment of CQ$^-$s is defined in the same manner as for positive ones: a CQ$^-$ $Q_1$ is contained in another one $Q_2$, denoted as $Q_1 \subseteq Q_2$, if for all databases $D$, $Q_1(D) \subseteq Q_2(D)$. Two CQ$^-$s are equivalent if and only if they are contained in each other.

3.2.2 The Containment Checking Algorithm for CQs

An algorithm for checking the containment of CQs was proposed in [CM77].

**Lemma 2 (Containment of CQs [CM77])** Consider two CQs $Q_1$ and $Q_2$:

$$Q_1 : h(\tilde{X}) := p_1(\tilde{X}_1), \ldots, p_n(\tilde{X}_n).$$

$$Q_2 : h(\tilde{U}) := q_1(\tilde{U}_1), \ldots, q_i(\tilde{U}_i).$$

Then $Q_1 \subseteq Q_2$ if and only if there exists a containment mapping $\rho$ from the variables of subgoals in $Q_2$ to those in $Q_1$, such that $\{\rho(q_1(\tilde{U}_1)), \ldots, \rho(q_i(\tilde{U}_i))\} \subseteq \{p_1(\tilde{X}_1), \ldots, p_n(\tilde{X}_n)\}$, and $\rho(h(\tilde{U})) = h(\tilde{X})$.

When the heads of both $Q_1$ and $Q_2$ do not contain variables, they are called boolean queries. It is obvious that boolean queries are the generalized forms of normal queries. The containment problem for CQs is shown to be NP-complete [CM77].

**Example 9 (Containment of CQs)** Consider the queries $Q_1$ and $Q_2$: the bodies of the queries are composed of the positive subgoals from Example 7.

$$Q_1 : q(X, Z) :- a(X, Y), a(Y, Z).$$

$$Q_2 : q(A, C) :- a(A, B), a(B, C), a(B, D).$$

There is one and only one containment mapping from $Q_2$ to $Q_1$:

$$\{A \rightarrow X, B \rightarrow Y, C \rightarrow Z, D \rightarrow Z\}.$$
3.3 Query Containment for $CQ^-$

In this section we discuss the containment checking for $CQ^-$. In the next subsection we introduce some necessary conditions.

3.3.1 Some Necessary Conditions

**Definition 6 (Super-Positive SP $Q^+$)** Given a $CQ^-$ $Q$ as follows:

$$Q: h(\bar{X}) := p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n), \neg s_1(\bar{Y}_1), \ldots, \neg s_m(\bar{Y}_m).$$

The SP of $Q$, denoted as $Q^+$ is: $h(\bar{X}) := p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)$.

**Lemma 3 (SP of $CQ^-$s)** Given a $CQ^-$ $Q$ with negated subgoals and its SP $Q^+$, $Q \sqsubseteq Q^+$ holds.

**Proposition 3 (Necessary condition for containment of $CQ^-$s)** Let $Q_1$ and $Q_2$ be two $CQ^-$, let $Q_1^+$ and $Q_2^+$ be their SP respectively. $Q_1 \sqsubseteq Q_2$ only if $Q_1^+ \sqsubseteq Q_2^+$.

**Proof 1** Assume $Q_1 \sqsubseteq Q_2$ and a tuple $t \in Q_1^+(D)$ where $D$ is any canonical database (i.e. each variable is assigned to a unique constant) of $Q_1^+$. We show that $t \in Q_2^+(D)$: Let $\rho$ be the substitution from variables of $Q_1$ to distinct constants in $D$. Let $s_i(\bar{Y}_i)(1 \leq i \leq m)$ be any negated subgoal in $Q_1$. Since $Q_1$ is satisfiable, therefore we obtain that $\rho(s_i(\bar{Y}_i)) \notin D$. Consequently, $t \in Q_1(D)$ and $t \in Q_2(D)$ are obtained. Following Lemma 3 it is obvious that $t \in Q_2^+(D)$. □

Proposition 3 provides a necessary but not sufficient condition for query containment of $CQ^-$. Next we give a theorem, stating a condition for $Q_1 \nsubseteq Q_2$.

**Theorem 4 (Necessary condition 2 for containment of $CQ^-$s)** Let $Q_1$ and $Q_2$ be two $CQ^-$. Assume $Q_1^+ \sqsubseteq Q_2^+$, and let $\rho_1, \ldots, \rho_r$ be all containment mappings from $Q_2^+$ to $Q_1^+$, such that $Q_1^+ \sqsubseteq Q_2^+$. $Q_1$ and $Q_2$ are given as follows:

$$Q_1: h(\bar{X}) := p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n), \neg s_1(\bar{Y}_1), \ldots, \neg s_m(\bar{Y}_m).$$

$$Q_2: h(\bar{U}) := q_1(\bar{U}_1), \ldots, q_i(\bar{U}_i), \neg a_1(\bar{W}_1), \ldots, \neg a_k(\bar{W}_k).$$

If for each $\rho_i(1 \leq i \leq r)$, there exists at least one $j(1 \leq j \leq k)$, such that $\rho_i(a_j(\bar{W}_j)) \in \{p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\}$, then $Q_1 \nsubseteq Q_2$. □
28.3. CONTAINMENT OF CONJUNCTIVE QUERIES WITH SAFE NEGATION

**Proof 2** A canonical database $D$ could be constructed as follows: freeze the positive subgoals of $Q_1$ and replace each variable in $Q_1$ with a distinct constant. We call this substitution $\sigma$. Let $t$ be any tuple such that $t \in Q_1(D)$, we have to show that $t \notin Q_2(D)$: that is, for each substitution $\theta$ which makes $t \in Q_2^+(D)$ true, there is at least one negated subgoal $a_j(W_j)$, where $1 \leq j \leq k$, such that $\theta(a_j(W_j)) \notin D$.

Since $p_1, \ldots, p_r$ are all the containment mappings from $Q_2^+$ to $Q_1^+$, it is true that $\theta \in \{\rho_1 \circ \sigma, \ldots, \rho_r \circ \sigma\}$. Assume $\theta = \rho_i \circ \sigma$ $(1 \leq i \leq r)$. Since for each $\rho_i$, there exists a $j (1 \leq j \leq k)$, such that $\rho_i(a_j(W_j)) \in \{p_\ell(X_\ell), \ldots, p_n(X_n)\}$, thus we have $\rho_1 \circ \sigma(a_j(W_j)) \in \{\sigma(p_1(X_1)), \ldots, \sigma(p_n(X_n))\}$ As a result, $\theta(a_j(W_j)) \notin D$ is obtained. □

### 3.3.2 Containment of CQ's

The following theorem states a necessary and sufficient condition for the containment checking of CQ's, which is one of the main contributions of this chapter.

**Theorem 5 (Containment of CQ's)** Let $Q_1$ and $Q_2$ be two CQ's as follows:

\[
Q_1: \quad h(X) \quad \vdash \quad p_1(X_1), \ldots, p_n(X_n), \neg s_1(Y_1), \ldots, \neg s_m(Y_m).
\]

\[
Q_2: \quad h(U) \quad \vdash \quad q_1(U_1), \ldots, q_l(U_l), \neg a_1(W_1), \ldots, \neg a_k(W_k).
\]

Then $Q_1 \subset Q_2$ if and only if

1. there is a containment mapping $\rho$ from $Q_2^+$ to $Q_1^+$ such that $Q_1^+ \subset Q_2^+$, and

2. for each $j (1 \leq j \leq k)$, $Q' \subset Q_2$ holds, where $Q'$ is as follows:

\[
Q': \quad h(X) \quad \vdash \quad p_1(X_1), \ldots, p_n(X_n), \neg s_1(Y_1), \ldots, \neg s_m(Y_m), \rho(a_j(W_j)).
\]

**Proof 3**

- $\Leftarrow$: Let $D$ be any database and $t$ the tuple such that $t \in Q_1(D)$, we have to prove that $t \in Q_2(D)$.

Since $t \in Q_1(D)$, we have immediately $t \in Q_1^+(D)$ and $t \in Q_2^+(D)$. Let $\sigma$ be the substitution from the variables in $Q_1^+$ to the constants in $D$ such that $t \in Q_1^+(D)$. Let $\theta = \rho \circ \sigma$. It is apparent that $\{\theta(q_1(U_1)), \ldots, \theta(q_l(U_l))\} \subset D$.

If for each $j (1 \leq j \leq k)$, $\theta(a_j(W_j)) \notin D$, then the result is straightforward. Otherwise, if there is any $j (1 \leq j \leq k)$, such that $\theta(a_j(W_j)) \in D$, then we have $\rho \circ \sigma(a_j(W_j)) \in D$. it can be concluded that $t \in Q'(D)$ where $Q'$ is

\[
Q': \quad h(X) \quad \vdash \quad p_1(X_1), \ldots, p_n(X_n), \neg s_1(Y_1), \ldots, \neg s_m(Y_m), \rho(a_j(W_j)).
\]

---

$^1$ $\rho \circ \sigma$ denotes the composition of substitutions $\rho$ and $\sigma$. 
3.3. QUERY CONTAINMENT FOR $CQ^{-}$S

From the assumption that $Q' \subseteq Q_2$ in the above theorem, $t \in Q_2(D)$ can then be obtained.

$\Rightarrow$: The proof is via deriving a contradiction.

1. If $Q_1'^+ \nsubseteq Q_2^+$, then from Proposition 3, $Q_1 \nsubseteq Q_2$ can be obtained immediately.
2. Otherwise, if for each containment mapping $\rho$ from $Q_2'^+ \rightarrow Q_1'^+$, such that $Q_1^+ \nsubseteq Q_2^+$, there is at least one $Q'$ as given in the above theorem, such that $Q' \nsubseteq Q_2$, then there exists at least one database $D$, such that $t \in Q'(D)$, but $t \notin Q_2(D)$. Since $Q'$ has only one more positive subgoal than $Q_1$, it is obvious that $t \in Q_1(D)$. This leads to the result that $Q_1 \nsubseteq Q_2$. \hfill \qed

Theorem 5 involves a recursive containment test. In each round, the containment $Q' \subseteq Q_2$ (the definition of $Q'$ see the above theorem) has to be verified. This might lead to one of the two results: (1) for each $Q'$, there is $Q' \subseteq Q_2$ — either $Q'$ is unsatisfiable, or via recursive containment test $\rightarrow$, then the test terminates with the result $Q_1 \subseteq Q_2$; (2) there exists a $Q'$, such that $Q' \nsubseteq Q_2$. This can be verified according to Theorem 4. In this case, the result of $Q_1 \nsubseteq Q_2$ can be obtained. The following example illustrates the algorithm. Note that we intentionally omit the variables in the head in order to generate more containment mappings from $Q_2'$ to $Q_1'$. It is not difficult to understand that the fewer containment mappings are there from $Q_2'$ to $Q_1'$, the simpler the test will be.

**Example 10 (Containment of $CQ^{-}$s)** Given the queries $Q_1$ and $Q_2$:

$$Q_1: \quad h \leftarrow a(X,Y), a(Y,Z), \neg a(X,Z).$$
$$Q_2: \quad h \leftarrow a(A,B), a(C,D), \neg a(B,C)$$

There are four containment mappings from $Q_2^+$ to $Q_1^+$, one of which is $\rho_1 = \{A \rightarrow Y, B \rightarrow Z, C \rightarrow X, D \rightarrow Y\}$. Now a new conjunctive query is generated as follows:

$$Q': \quad h \leftarrow a(X,Y), a(Y,Z), a(Z,X), \neg a(X,Z).$$

Note that the subgoal $a(Z,X)$ is generated from $\rho_1(a(B,C))$. One of the containment mappings from $Q_2$ to $Q'$ is $\rho_2 = \{A \rightarrow Z, B \rightarrow X, C \rightarrow Z, D \rightarrow X\}$. Since the newly generated subgoal $\rho_2(a(B,C))$ is $a(X,Z)$, this leads to a successful unsatisfiability test of $Q''$.

$$Q'': \quad h \leftarrow a(X,Y), a(Y,Z), a(Z,X), a(X,Z), \neg a(X,Z).$$

it can then be concluded that $Q' \subseteq Q_2$. In the sequel we have $Q_1 \subseteq Q_2$. 

The detailed algorithm is given at the end of this chapter. The idea behind the
algorithm can be informally stated as follows: we start with the positive subgoals
of $Q_1$ as root. Let $r$ be the number of all containment mappings from $Q_2^+$ to $Q_1^+$,
such that $Q_1^+ \subseteq Q_2^+$. $r$ branches are generated from the root, with sets of mapped
negated subgoals as nodes (cf. Figure 3.1(a)). Next, each node might be marked
as Contained, if it is identical to one of the negated subgoals of $Q_1$, or as Terminal,
if it is identical to one of the positive subgoals of $Q_1$. If there exists one branch
such that each node is marked as Contained, then the program terminates with
the result $Q_1 \subseteq Q_2$. Otherwise, if at least one node of each branch is marked as
Terminal, then the program terminates too, however with the result $Q_1 \not\subseteq Q_2$.
If none of these conditions is met, the program continues with expansion of the
non-terminal nodes, that is, the nodes mark with Terminal will not be expanded
any more.

It can be shown that the algorithm terminates. The reasons are: (1) the exp-
ansion does not generate new variables; (2) the number of variables in $Q_1$ is finite.

The next example shows how the algorithm terminates when the complement
problem $Q_1 \not\subseteq Q_2$ is solved.

Example 11 (Containment of $CQ^-$s) Given the queries $Q_1$ and $Q_2$:


In Figure 3.1, it is shown that there are four containment mappings from $Q_2^+$ to
$Q_1^+$: $\rho_1, \ldots, \rho_4$. Each mapping contains two sub-trees since there are two negated
subgoals in $Q_2$. The branches $\rho_2$ and $\rho_3$ are marked as Terminal because there is
at least one Terminal node from each of the above branches (note that we denote a
node Terminal with a shadowed box around it, and Contained with a normal box).
The node $a(X, Z)$ from branch $\rho_1$ is marked as Contained, because it is the same
as the negated subgoal in $Q_1$. (Note that in Figure 3.1 the node $a(Y, Y)$ of branch
$\rho_1$ is marked as Terminal, but it is the result of the next round. Up to now, it has
not been marked. The same holds for the nodes of branch $\rho_4$.)

Next the non-terminal node $a(Y, Y)$ is expanded. Five new containment mappings are generated as $\rho_5, \ldots, \rho_9$. Since all the branches are Terminal, and $a(Y, Y)$
is also a sub-node of $\rho_4$, it can be concluded that the expanded query

$$Q': h \leftarrow a(X, Y), a(Y, Z), a(Y, Y), \neg a(X, Z).$$

is not contained in $Q_2$. Because all the containment mappings from $Q_2^+$ to $Q'^+$
have been verified. As a result, $Q_1 \not\subseteq Q_2$ is obtained.
3.3. QUERY CONTAINMENT FOR CQ’S

Comparison of the algorithms. We notice several interesting similarities and
differences between our algorithm and the one in [Ull97]: The partitioning of vari-
ables is similar to the step of checking of $Q_1^+ \subseteq Q_2^+$. It can be proven that if the
containment checking $Q_1^+ \subseteq Q_2^+$ is successful, there exists at least one partition
of variables, namely the partition with a distinct constant for each variable, such
that when applied to the canonical database built from this partition, $Q_2$ yields
the same answer set as $Q_1$. As a matter of fact, we do not see the necessities of
checking/expanding other partitions as the one with \{X, Z\}, \{Y\} in Table 3.1, at
least concerning the algorithm in this chapter.

The next step of the algorithm in [Ull97] is to check an exponential number of
canonical databases, as described in the introduction. In contrast, our algorithm
continues with the containment mappings of their positive counterparts and execu-
tes the specified test, which takes linear time in the size of $Q_1$. If the test is
not successful, the query is extended with one more positive subgoal (but without
new variables) and the next containment test continues. It is important to empha-
size that in the worst case, the expanded tree generates all the nodes composed

![Diagram of generated tree and containment mappings]

Figure 3.1: The graphic illustration of Example 11.
of variant combinations of the variables in $Q_1$, which coincides with the method in [Ull97].

However, in the following examples we show that our algorithm terminates definitely earlier in the some cases:

(1) It turns out that the result of the containment test is $Q_1 \subseteq Q_2$. As explained in the Introduction, the algorithm in [Ull97] terminates if one canonical database as counter-example is found, but in order to obtain the result of $Q_1 \subseteq Q_2$, all canonical databases have to be verified. In contrast, our algorithm terminates with the result $Q_1 \subseteq Q_2$ if the specified test is successful.

**Example 12 (Containment of CQ’s)** Given the queries $Q_1$ and $Q_2$ as in Example 7. There is a containment mapping $\rho$ from $Q_2^+$ to $Q_1^+$ as in Example 9. There is only one $Q'$ to be checked:

$$Q': \ q(X, Z) \rightarrow a(X, Y), a(Y, Z), \neg a(X, Z), a(X, Z).$$

It is apparent that $Q'$ is unsatisfiable, thus we have proven that $Q_1 \subseteq Q_2$.

(2) To test $Q_1 \subseteq Q_2$, if none of the predicates from negated subgoals in $Q_2$ appears in positive subgoals, the test terminates after the first round of containment checking of $Q' \subseteq Q_2$. It can be explained as follows: due to the assumption, the containment mappings from $Q_2$ to the newly generated query $Q'$ (see Theorem 5) are the same as the ones from $Q_2$ to $Q_1$. Since no new containment mapping is generated, the algorithm reaches some "fix-point", so that no new branches of the expanded tree will be generated.

**Example 13 (Containment of CQ’s)** Given the queries $Q_1$ and $Q_2$ as in Example 8. There is a containment mapping $\rho_1$ (cf. Figure 3.2). A new node $\rho_1(b(C, C)) = b(Z, Z)$ is then generated. The new query $Q'$ is the following:

$$Q': \ q(X, Z) \rightarrow a(X, Y), a(Y, Z), \neg a(X, Z), b(Z, Z).$$

Since $\rho_1$ is the only containment mapping from $Q_2$ to $Q'$, we mark the node $b(Z, Z)$ as Terminal. Following Theorem 4, $Q' \not\subseteq Q_2$ can be obtained, which is followed by the result $Q_1 \not\subseteq Q_2$.

At last, it should be mentioned that the algorithm in [Ull97] can deal with unsafe negations, while ours cannot.
3.4. CONTAINMENT OF UNIONS OF $CQ$'s

\[
\begin{align*}
\text{a}(X,Y), \text{a}(Y,Z) & \quad \mid \rho_1 \\
\text{b}(Z,Z) & \quad \rho_1 \{ A \rightarrow X, B \rightarrow Y, C \rightarrow Z \}
\end{align*}
\]

(a) The generated tree \hspace{1cm} (b) The containment mapping

Figure 3.2: The graphic illustration of Example 13.

3.4 Containment of Unions of $CQ$’s

In this section we consider the containment problem of unions of $CQ$’s. First we present some basic notations.

However, when negation is allowed in $CQ$s, the above theorem will not hold any more.

**Example 14 (Containment of unions of $CQ$’s)** Consider the following queries with the relations man, father, and husband:

\[
\begin{align*}
Q : \quad q(X) & \quad \leftarrow \text{man}(X), \neg\text{father}(X). \\
Q_1 : \quad q(Y) & \quad \leftarrow \text{man}(Y), \neg\text{husband}(Y). \\
Q_2 : \quad q(Z) & \quad \leftarrow \text{man}(Z), \text{husband}(Z), \neg\text{father}(Z).
\end{align*}
\]

It is apparent that neither $Q \subseteq Q_1$, nor $Q \subseteq Q_2$, but $Q \subseteq Q_1 \cup Q_2$.

The containment of unions of $CQ$’s is defined in the similar way as in Definition 5. The following notations have to be additionally presented.

**Definition 7 (Unions of $CQ$’s)** Let $Q = Q_1 \cup \ldots \cup Q_n$ be a union of $CQ$’s. $Q^+ = Q_1^+ \cup \ldots \cup Q_n^+$.

**Lemma 4 (Unions of $CQ$’s)** Let $Q = Q_1 \cup \ldots \cup Q_n$ be a union of $CQ$’s. $Q \subseteq Q^+$ holds.

**Lemma 5 (Containment of unions of $CQ$’s)** Given a $CQ$ $Q$ and a union of $CQ$’s $Q = Q_1 \cup \ldots \cup Q_n$. If $Q \subseteq Q$, then there exists at least one $Q_i \in Q(1 \leq i \leq n)$, such that $Q^+ \subseteq Q_i^+$.

**Proof 4** (sketch) Using the same method as in Proposition 3, we have $Q^+ \subseteq Q_1^+ \cup \ldots \cup Q_n^+$. Following [SY80] the result can be obtained. \qed
343. **CONTAINMENT OF CONJUNCTIVE QUERIES WITH SAFE NEGATION**

**Theorem 6 (Containment of unions of CQ^-s)** Given a CQ^- Q and a union of CQ^-s \( Q = Q_1 \cup Q_2 \cup \ldots \cup Q_v \). Assume \( Q^+ \subseteq Q^+ \), and let \( \rho_{i,1}, \ldots, \rho_{i,r_i} \) (for \( i = 1, \ldots, v \)) be all the containment mappings from \( Q^- \) to \( Q^+ \), such that \( Q^+ \subseteq Q^+ \). Q and \( Q_1, \ldots, Q_v \) are given as follows:

\[
Q : h(X) \leftarrow p_1(X_1), \ldots, p_n(X_n), \neg s_1(Y_1), \ldots, \neg s_m(Y_m).
\]

\[
Q_i (1 \leq i \leq v) : h(U) \leftarrow q_{i,1}(U_{i,1}), \ldots, q_{i,t_i}(U_{i,t_i}), \neg a_{i,1}(W_{i,1}), \ldots, \neg a_{i,k_i}(W_{i,k_i}).
\]

If for each \( \rho_{i,j} (1 \leq i \leq v, 1 \leq j \leq r_i) \), there exists at least a \( a_{i,u}(W_{i,u}) \), where \( 1 \leq u \leq k_i \), such that \( \rho_{i,j}(a_{i,u}(W_{i,u})) \in \{ p_1(X_1), \ldots, p_n(X_n) \} \), then \( Q \nsubseteq Q \).

**Proof 5 (sketch)** A canonical database \( D \) can be built in the similar way as in Theorem 4. Then it can be proven that a tuple \( t \in Q_1(D) \), but \( t \notin Q_2(D) \).

The following theorem states a sound and complete condition.

**Theorem 7** Let \( Q \) be a CQ^- and \( Q \) be the union of CQ^-s \( Q = Q_1 \cup Q_2 \cup \ldots \cup Q_v \) as follows:

\[
Q : h(X) \leftarrow p_1(X_1), \ldots, p_n(X_n), \neg s_1(Y_1), \ldots, \neg s_m(Y_m).
\]

\[
Q_i (1 \leq i \leq v) : h(U) \leftarrow q_{i,1}(U_{i,1}), \ldots, q_{i,t_i}(U_{i,t_i}), \neg a_{i,1}(W_{i,1}), \ldots, \neg a_{i,k_i}(W_{i,k_i}).
\]

Then \( Q \subseteq Q \) if and only if

- there is a containment mapping \( \rho \) from \( Q^+_u \) to \( Q \), where \( 1 \leq u \leq v \), such that \( Q^+ \subseteq Q^+ \), and

- for each \( j (1 \leq j \leq k_u) \), \( Q' \subseteq Q \) holds, where \( Q' \) is as follows:

\[
Q' : h(X) \leftarrow p_1(X_1), \ldots, p_n(X_n), \neg s_1(Y_1), \ldots, \neg s_m(Y_m), \rho(a_{u,j}(W_{u,j})).
\]

The algorithm behind the theorem can be adapted with little modification from that of containment checking for two CQ^-s. Next we explain how the containment checking of Example 14 is executed.

**Example 15 (Containment of unions of CQ^-s)** Consider the queries \( Q, Q_1 \) and \( Q_2 \) in Example 14. Since \( Q_1 \) is the only query whose positive part of body contains that of \( Q \), the containment mapping \( \rho_1 = \{ Y \rightarrow X \} \) is obtained. The newly generated \( Q' \) has the following form:

\[
Q' : q(X) \leftarrow \text{man}(X), \neg \text{father}(X), \text{husband}(X).
\]

Since it is easy to check that \( Q' \subseteq Q_2 \), it can then be concluded that \( Q \subseteq Q_1 \cup Q_2 \).
3.5 Some Notes on Answering Queries using Views

The problem of answering queries using views has been intensively studied recently, especially in the area of data integration (see [Lev00] for a survey). The problem can be described as follows: given a query \( Q \) over a database schema, and a set of view definitions \( \mathcal{V} \) over the same schema, how to find a rewriting \( Q' \) of \( Q \), using only the views. The notations of equivalent rewriting and maximally-contained rewriting can be found in [Lev00].

Due to the close relation between the problem of query containment and answering queries using views, we are interested in giving the discussion of some new issues for answering queries using views, when safe negation is allowed in both queries and views. Since the completeness of any rewriting algorithm depends on the query language in which the rewritings are expressed, it has to be specified whether queries with unions and negations are allowed for expressing the rewriting. In the following example, we show that even without applying negation on the views, the equivalent rewriting can only be found if unions are allowed in the rewriting.

**Example 16 (Query rewriting with unions of \( \text{CQ}^- \)s)** Let \( Q \) and \( V_1, V_2 \) be the conjunctive queries as follows:

\[
Q : \quad q(X, Y) \quad : \quad a(X, Y).
\]

\[
V_1 : \quad v_1(X, Y) \quad : \quad a(X, Y), b(Y).
\]

\[
V_2 : \quad v_2(X, Y) \quad : \quad a(X, Y), \neg b(Y).
\]

There is an equivalent rewriting of \( Q \) using the union of \( V_1 \) and \( V_2 \).

One important issue concerning answering queries using views is to decide whether a view is useful for the given query. It can be informally stated as the following [Lev00]: a view can be useful for a query if the set of relations it mentions overlaps with that of the query, and it selects some of the attributes selected by the query. When a query with both positive and negative subgoals is considered, this condition remains true for the positive subgoals, but not for the negative subgoals any more. It can be shown in the following example that an equivalent rewriting can only be found by negating the views.

**Example 17 (Query rewriting with unions of \( \text{CQ}^- \)s)** Let \( Q \) and \( V_1, V_2 \) be the conjunctive queries as follows:

\[
Q : \quad q(X, Y) \quad : \quad p_0(X, Y), \neg p_1(X, X).
\]

\[
V_1 : \quad v_1(X, Y) \quad : \quad p_0(X, Y).
\]

\[
V_2 : \quad v_2(X, Z) \quad : \quad p_0(X, Y), p_1(X, Z).
\]
36.3. CONTAINMENT OF CONJUNCTIVE QUERIES WITH SAFE NEGATION

It can be shown that the rewriting $Q'$ is an equivalent one:

$$Q' : q(X, Y) :\dashv v_1(X, Y), \neg v_2(X, X).$$

One difficulty in choosing the views for the negated subgoals is to decide whether a view $V$ can be useful for the query $Q$ with its negated form. The next example differs from Example 17 only on $V_2$.

**Example 18 (Query rewriting with unions of $CQ^\neg$'s)** Let $Q$ and $V_1$, $V_2$ be the conjunctive queries as follows:

$$Q : q(X, Y) :\dashv p_0(X, Y), \neg p_1(X, X).$$
$$V_1 : v_1(X, Y) :\dashv p_0(X, Y).$$
$$V_2 : v_2(X, Z) :\dashv p_1(X, Z), p_2(X, Y).$$

The rewriting $Q'$ as follows is not even a contained rewriting for $Q$.

$$Q' : q(X, Y) :\dashv v_1(X, Y), \neg v_2(X, X).$$

When $Q' \cup V_1 \cup V_2$ is applied to the database $D = \{p_0(0, 1), p_1(0, 0)\}$, tuple $q(0, 1)$ is obtained. However it is apparent that this is not the correct answer for $Q$.

Informally speaking, in order to be useful for the query $Q$ with its negated form, a view $V$ should not contain foreign predicates, which do not appear in $Q$ (cf. Example 18). Moreover, $V$ has to contain at least one positive subgoal whose negated form appears in $Q$, and the attributes of the subgoal should not be projected out.

**Related Work** There is relatively less research concerning the problem of answering queries using views with safe negations appearing in the body of both queries and views. [FG91] provides an algorithm dealing with it. It is an extension of the inverse rule algorithm. However it is not clear whether the algorithm is complete w.r.t the problem of views based query answering.

### 3.6 Conclusion and Further Research

We have discussed the query containment problem for the conjunctive queries with safe negated subgoals. A new method for testing the containment of $CQ^\neg$'s is given, comparing with the one in [Ul97]. We have also shown that this algorithm can be naturally extended to the containment checking of unions of $CQ^\neg$'s. At last,
we have discussed the problem of answering queries using views when both queries and views have negated subgoals. Some motivating examples are given to show the cases which might not be encountered with pure conjunctive queries.

There are several interesting questions left for research. In [AD98] several categories of complexities for answering queries using views are given. However, with respect to negation, Datalog\textsuperscript{\neg} is considered, and the setting of CQ\textsuperscript{\neg} is not covered.

One interesting extension of our work is to consider the containment problem of CQs with both safe negation and arithmetic comparisons. To the best of our knowledge, no practical algorithm has been published to solve this problem. The complexity of this problem is still unknown as well.

APPENDIX

Algorithm

**Containment Checking**($Q_1, Q_2$)

**Inputs:** $Q_1$ and $Q_2$ are CQ\textsuperscript{\neg}s with the form as follows:

\begin{align*}
Q_1 & : \quad h(\bar{X}) \leftarrow p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n), \neg s_1(\bar{Y}_1), \ldots, \neg s_m(\bar{Y}_m). \\
Q_2 & : \quad h(\bar{U}) \leftarrow q_1(\bar{U}_1), \ldots, q_l(\bar{U}_l), \neg a_1(\bar{W}_1), \ldots, \neg a_k(\bar{W}_k).
\end{align*}

**Begin**

1. Set \{\(p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\}\} as the root of a tree: \(c_0\).
   Let \(\rho_1, \ldots, \rho_r\) be all the containment mappings from \(Q_2^+\) to \(Q_1^+\).
2. Generate \(r\) nodes \(c_1, \ldots, c_r\) as children of root, and each node \(c_i(1 \leq i \leq r)\) has a subtree with \(k\) children, in which each child \(c_{i,j}\) with the form \(\rho_i(a_j(\bar{W}_j))(1 \leq j \leq k)\).
3. Marking the nodes:
   For \(i=1\) to \(r\) do
     For \(j=1\) to \(k\) do
       If \(c_{i,j} \in \{p_1(\bar{X}_1), \ldots, p_n(\bar{X}_n)\}\) Then mark \(c_i\) as *Terminal*;
       If \(c_{i,j} \in \{s_1(\bar{Y}_1), \ldots, s_m(\text{ymbar})\}\) Then mark \(c_{i,j}\) as *Contained*;
     EndFor
   EndFor
4. Execute the containment checking:
   If all nodes \(c_1, \ldots, c_r\) are terminal nodes, Then return *Not-contained*;
   For \(i=1\) to \(r\) do
     If each \(c_{i,j}(1 \leq j \leq k)\) is marked as *Contained*, Then return *Contained*;
   EndFor
38.3. \textit{CONTAINMENT OF CONJUNCTIVE QUERIES WITH SAFE NEGATION}

EndFor

5. Continue expanding non-terminal nodes:

For i=1 to r do

If $c_i$ is not \textit{Terminal} Then

For j = 1 to k do

If $c_{i,j}$ is not \textit{Contained} Then

let $Q'_1$ be: $h :: p_1(\tilde{X}_1), \ldots, p_n(\tilde{X}_n), c_{i,j}, \neg s_1(\tilde{Y}_1), \ldots, \neg s_m(\tilde{Y}_m)$;

let $\rho_{i,j,1}, \ldots, \rho_{i,j,b_{i,j}}$ be the new containment mappings from $Q_2$ to $Q'_1$;

Generate $b_{i,j}$ nodes with children in the same way as in \textbf{Step 2};

Mark the nodes in the same way as \textbf{Step 3};

EndIf

EndFor

EndIf

EndFor

EndIf

6. Execute the containment checking:

For i=1 to r do

If each $c_{i,j}(1 \leq j \leq k)$ either is marked as \textit{Contained}

or has a child whose children are all marked \textit{Contained}

Then return \textit{Contained};

EndIf

from each non-terminal node $c_i$, choose one child $c_{i,j}(1 \leq j \leq k)$;

add all these $c_{i,j}$ to the body of $Q_1$, and let the new query be $Q''_1$;

If each branch w.r.t. $Q''_1$ is marked as \textit{Terminal}, Then return \textit{Not-contained};

EndFor

Let $r$ be all expanded nodes; \textbf{Go to} Step 5;

End
4 CONTAINMENT OF ACYCLIC CONJUNCTIVE QUERIES

4.1 Introduction

This chapter considers the problem of query containment of acyclic conjunctive queries with safe negated subgoals. The query containment problem is to check whether the answer set of one query is a subset of another query for all databases. Algorithms for query containment are of interest in several contexts in the database area.

A conjunctive query is called acyclic, if the hypergraph of the query is acyclic [AHV95]. The notion of the acyclic hypergraph was initially introduced in query optimization. It is shown that if the hypergraph of a conjunctive query is acyclic, then a sequence of semi-join operations can be conducted with pair-wise conjuncts, to delete the "dangling tuples", before the join algorithm is executed. In fact, Yannakakis [Yan81] showed that query evaluation of acyclic CQs is in polynomial time.

A recent result from Gottlob et al. [GLS01] shows that it is LOGCFL complete, which allows highly parallelizable query processing.

Recently, there is renewed interest in acyclic CQs and CQs with bounded tree width, which is a general form of acyclicity. It is originated from two active research communities: (1) in model checking, guarded fragment of first-order logic is proposed as a general form of modal logic, to explain the nice properties of several temporal logics such as decidability of the temporal logics and linear time model checking of CTL. There is a natural correlation between guarded FO and acyclic CQ [FFG02]. (2) XPath, the query model for XML, is in essence acyclic. In fact, XPath is based on navigation through the XML tree by path expression of the form //step/step/.../step, in which each single step is of the form axis::nodetest[filter]*. At each step, the current node is only related with the node before it, and the one after it. This property explains (partly) the linear query evaluation of XPath algorithm proposed recently [GK02a].
In this chapter, we consider the containment problem of acyclic conjunctive queries with safe negation. Firstly, we give a syntactical definition of acyclic conjunctive queries with safe negation. Secondly, we show that the problem is in Co-NP, by an algorithm adapted directly from Ullman’s algorithm. At last, we prove that the problem is Co-NP hard, by a reduction from the co-problem of 3SAT. The proof shows that under the universe \{0, 1\}, the containment checking is Co-NP hard. The hardness proof for the containment checking problem under any universe is left open.

4.2 Preliminaries

We first define acyclic \(CQ^-\), which is adapted from Flum et al. [FFG02].

**Definition 8 (Acyclic \(CQ^-\))** Given a \(CQ^-\) \(Q\), the hypergraph \(H(Q)\) is obtained by including both positive and negative atoms as vertices. A \(CQ^-\) \(Q\) is acyclic if

1. the hypergraph \(H(Q)\) is acyclic, and
2. there is a corresponding join tree in which all negated atoms appear only on leaves.

Note that the definition is equivalent to that given by Flum et al. ([FFG02] Definition 5.17), which requires that for every negative atom \(\alpha\), there exists a positive atom \(\beta\), such that \(\text{var}(\alpha) \subseteq \text{var}(\beta)\). According to this definition, if any atom \(\gamma\) appears as the child of \(\alpha\), it is also the child of \(\beta\), so that there is a join tree in which negative atoms appear only on leaves.

**Example 19 (Acyclic \(CQ^-\))** Consider the following queries \(Q_1\) and \(Q_2\),

\[
Q_1 : q() \ :- \ r(A, B), r(B, C), \neg r(C, A).
\]

\[
Q_2 : q() \ :- \ r(A, B), r(B, C), \neg r(C, C).
\]

It is obvious that \(Q_1\) is cyclic, while \(Q_2\) is acyclic.

4.2.1 Query evaluation and containment for acyclic \(CQs\)

In Table 4.1 an overview is given for various complexity results of query evaluation and containment for \(CQs\), \(CQ^-s\), acyclic \(CQs\), and acyclic \(CQ^-s\) respectively. The bottom-up polynomial time algorithm of query evaluation for acyclic \(CQs\) and \(CQ^-s\) is not hard to understand. Given a join tree of any query, we can start from leaves and execute semi-joins between each leaf and its parent. After all the leaves are processed, the parent itself becomes a leaf, and this operation can be performed till the root.
4.3 Query containment for acyclic $CQ^-$s

In this section we discuss the containment checking for acyclic $CQ^-$s. First, we show that the containment checking problem of $Q_1 \subseteq Q_2$, is in Co-NP, where $Q_1$ and $Q_2$ are $CQ^-$s and $Q_2$ is acyclic. Further, we prove that problem is also Co-NP hard, by a reduction from the 3SAT problem.

### 4.3.1 Upper bounds

Ullman [Ull97] proposed an algorithm based on the approach of canonical databases by testing an exponential number of canonical databases. Example 7 in Chapter 3 explains the approach.

The algorithm takes exponential numbers of query evaluation on canonical databases. Since the query evaluation of $CQ^-$ is in NP, it can be concluded that the containment test is in $\Pi^p_2$.

**Proposition 4 (Complexity of $CQ^-$ containment [Ull97])** Given two $CQ^-$s $Q_1$ and $Q_2$, the containment test of $Q_1 \subseteq Q_2$ is in $\Pi^p_2$. □

Consider the containment checking of acyclic $CQ$-s, the same algorithm can be applied. Exponential numbers of canonical databases have to be evaluated. However, the query evaluation of acyclic $CQ^-$ is in PTIME. It follows naturally that the containment test is in Co-NP.

**Proposition 5 (Containment of acyclic $CQ^-$ containment, upper bound)** Given two $CQ^-$s $Q_1$ and $Q_2$, where $Q_2$ is acyclic. The containment test of $Q_1 \subseteq Q_2$ is in Co-NP. □

### 4.3.2 Lower bounds

In this section, we show that the Co-NP upper bound is tight.
4. CONTAINMENT OF ACYCLIC CONJUNCTIVE QUERIES

Theorem 8 (Containment of acyclic \( CQ^\sim \) containment, lower bound)

Given two \( CQ^\sim \)'s \( Q_1 \) and \( Q_2 \), where \( Q_2 \) is acyclic. The containment test of \( Q_1 \subseteq Q_2 \) is Co-NP hard.

Proof 6 We reduce the co-problem of 3SAT, which is known to be Co-NP complete, to our problem. The 3SAT problem is defined as follows: Given a 3SAT formula \( F \) with variables \( \bar{x} \), is there any truth assignment to \( \bar{x} \) that satisfies \( F \)?

We are given a 3SAT formula \( F \), with variables \( \bar{x} = \{x_1, \ldots, x_n\} \) and clauses \( \mathcal{C} = \{c_1, \ldots, c_p\} \). Clause \( c_i \) contains the three variables (either positive or negated) \( z_{i,1}, z_{i,2}, \) and \( z_{i,3} \).

We now construct two acyclic \( CQ^\sim \)'s \( Q_1 \), and \( Q_2 \). The queries are defined over the predicates \( e \) and \( r_1, \ldots, r_p \), where \( e \) is with arity \( n \) and \( r_1, \ldots, r_p \) are with arity \( 3 \). The query \( Q_2 \) is defined as follows:

\[
Q_2 : \; q() := e(x_1, \ldots, x_n), \neg r_1(z_{1,1}, z_{1,2}, z_{1,3}), \ldots, \neg r_p(z_{p,1}, z_{p,2}, z_{p,3}).
\]

It is obvious that \( Q_2 \) is acyclic.

\( Q_1 \) is constructed as follows: for each clause \( c_i \) in \( F \), there is one truth assignment \( \{a_{i,1}, a_{i,2}, a_{i,3}\} \) to \( \{z_{i,1}, z_{i,2}, z_{i,3}\} \), that does not satisfy \( c_i \). The body of \( Q_1 \) contains the following subgoals:

\[
Q_1 : \; q() := e(x_1, \ldots, x_n), r_1(z_{1,1}, z_{1,2}, z_{1,3}), \ldots, r_p(z_{p,1}, z_{p,2}, z_{p,3})
\neg r_1(a_{1,1}, a_{1,2}, a_{1,3}), \ldots, \neg r_p(a_{p,1}, a_{p,2}, a_{p,3}).
\]

For example, consider the formula

\[
(x_1 \lor \bar{x}_2 \lor x_3) \land (x_1 \lor x_2 \lor \bar{x}_4)
\]

the \( CQ^\sim \)'s \( Q_1 \) and \( Q_2 \) are as follows. Note that we rename the variables in \( Q_1 \) and \( Q_2 \) to avoid symbolic confusion.

\[
Q_1 : \; q() := e(u_1, u_2, u_3, u_4), r_1(u_1, u_2, u_3), r_2(u_1, u_2, u_4), \neg r_1(0, 1, 0), \neg r_2(0, 0, 1).
\]

\[
Q_2 : \; q() := e(v_1, v_2, v_3, v_4), \neg r_1(v_1, v_2, v_3), \neg r_2(v_1, v_2, v_4).
\]

In the following we prove that the formula \( F \) is satisfiable if and only if \( Q_1 \not\subseteq Q_2 \).

- First we prove that if \( F \) is satisfiable, then \( Q_1 \not\subseteq Q_2 \). Since \( F \) is satisfiable, there exists one truth assignment \( \{a_1, \ldots, a_n\} \) to \( \{x_1, \ldots, x_n\} \), such that every clause \( c_i \) is satisfiable. Let \( b_{i,1}, b_{i,2}, b_{i,3} \) be the corresponding assignments of \( z_{i,1}, z_{i,2}, z_{i,3} \) (1 \( \leq \) \( i \) \( \leq \) \( p \)) respectively. The database \( D \) is constructed as follows:

\[
D : \{ e(a_1, \ldots, a_n), r_1(b_{1,1}, b_{1,2}, b_{1,3}), \ldots, r_p(b_{p,1}, b_{p,2}, b_{p,3}) \}
\]

It is obvious that \( q \in Q_1(D) \). However if \( Q_2 \) is applied to \( D \), we can obtain that \( q \notin Q_2(D) \). As a result, we have \( Q_1 \not\subseteq Q_2 \).
4.4. CONCLUSION

- Next we show that if $F$ is not satisfiable, then $Q_1 \subseteq Q_2$. If $F$ is not satisfiable, then there does not exist any truth assignment to $\{x_1, \ldots, x_n\}$, such that every clause $c_i$ is satisfiable. This means, for every possible assignment, there is at least one clause $c_i (1 \leq i \leq p)$, which is not satisfiable. Note that in $Q_1$, the truth assignments making every clause unsatisfiable are recorded in negated subgoals. This follows that over the universe $\{0, 1\}$, there does not exist a database $D$, such that $Q_1(D)$ is true. Since $Q_1$ is not satisfiable, it follows that $Q_1 \subseteq Q_2$ always holds. So the theorem follows.

Remark Note that the hardness proof succeeds only over the universe $\{0, 1\}$. Whether it succeeds over any universe is unknown.

4.4 Conclusion

We have discussed the query containment problem for the acyclic conjunctive queries with safe negated subgoals. We showed that the problem is Co-NP complete, by giving both the upper and lower bounds. The proof shows that under the universe $\{0, 1\}$, the containment checking is Co-NP hard. The hardness proof for the containment checking problem under any universe is left open.
4. CONTAINMENT OF ACYCLIC CONJUNCTIVE QUERIES
5

CONTAINMENT OF
CONJUNCTIVE QUERIES
WITH DISJUNCTION

5.1 Introduction

In this chapter, we provide an algorithm for containment checking problem under disjunctive referential constraints.

The query containment problem in the presence of integrity constraints is studied first in [JK82], especially with functional and inclusion dependencies. In [ZÖ97], the integrity constraints are extended to implication constraints and referential constraints, which are the generalized form of functional constraints and inclusion dependencies, respectively. However, to handle incomplete information in the database, disjunctions are needed to be expressed as integrity constraints. The following example illustrates a situation where incomplete information gives rise to disjunctive integrity constraints.

Example 20 (A relational database schema) Consider the following database:

1. The relation schemes:
   \[ \text{empl} \text{(worker\_name)},  \text{group} \text{(group\_name)}, \text{member} \text{(worker, group\_name, work\_name)},  \text{same\_skill} \text{(worker, worker)}. \]

2. The following disjunctive constraint states that each employee belongs to one of the two groups, \( p_1 \) and \( p_2 \).
   \[
   \text{drc: } \text{empl}(X) \rightarrow \exists W_1 \text{ member}(X, p_1, W_1) \lor \exists W_2 \text{ member}(X, p_2, W_2). 
   \]

3. The implication constraint states that if two workers have the same skill, then they should not belong to the same group.
   \[
   \text{ic1: } \text{same\_skill}(X, X) \rightarrow . \\
   \text{ic2: } \text{empl}(X), \text{empl}(Y), \text{same\_skill}(X, Y), \text{member}(X, G, W_1), \text{member}(Y, G, W_2) \rightarrow . 
   \]
4. Finally, a relation \( q \) to define all pairs of employees working in different groups is composed as the union of two queries, denoted \( Q \), of \( Q_1 \) and \( Q_2 \):

\[
Q_1 : \quad q(X,Y) \leftarrow \text{empl}(X), \text{empl}(Y), \text{member}(X,p_1,W_1), \text{member}(Y,p_2,W_2).
\]

\[
Q_2 : \quad q(X,Y) \leftarrow \text{empl}(X), \text{empl}(Y), \text{member}(X,p_2,W_3), \text{member}(Y,p_1,W_4).
\]

Now we ask the query "list all the two-worker pairs who have the same skills":

\[
Q : \quad q(X,Y) \leftarrow \text{empl}(X), \text{empl}(Y), \text{same\_skill}(X,Y).
\]

Assuming the constraints have been enforced, that is, only relation instances satisfying \( \text{drc} \) and \( \text{ic1, ic2} \) are stored in the database, then we can get that \( Q \) is contained in the union of \( Q_1 \) and \( Q_2 \), i.e. \( Q \subseteq Q_1 \cup Q_2 \). From the disjunctive constraint we know, that every employee must work in \( p_1 \) or \( p_2 \). The implication constraints then enforce, that employees having same skills cannot work in the same group. Therefore, any answer to our query \( Q \) is guaranteed to be an element of the union \( Q \). However, without the constraints enforced, this containment relationship would not hold any more.

Moreover, in dealing with the query language like XPath for XML, disjunction is proposed in integrity constraints expressing the schema information of XML [DT01]. The next integrity constraint is taken from [DT01], stating that if a node \( u \) is the descendant of both \( x \) and \( y \), then either \( x \) and \( y \) are the same node, or \( x \) is the descendant of \( y \), or \( y \) is the descendant of \( x \).

\[
(\text{line}) \quad \forall x, y, u \quad [\text{desc}(x,u) \land \text{desc}(y,u) \rightarrow x = y \lor \text{desc}(x,y) \lor \text{desc}(y,x)]
\]

In this chapter we introduce disjunctive referential integrity constraints and give a sound and complete algorithm for checking the containment of conjunctive queries under disjunctive referential and implication constraints. The technique for handling disjunctive referential constraints is related to the well known minimal model semantics for disjunctive logic programming [LMR92]. Our work generalizes the results of [ZÖ97] in which only referential constraints without disjunctions are considered; we give a solution to a question left open so far.

### 5.2 Preliminaries

**Definition 9 (Implication Constraint)** [ZÖ97] An implication constraint is a formula of the form

\[
p_1(Y_1), ..., p_n(Y_n), I \rightarrow .
\]

where \( p_1(Y_1), ..., p_n(Y_n) \) are atoms and \( I \) is a conjunction of inequalities as defined above. Note that implication constraints are called denial constraints too.
5.3. QUERY CONTAINMENT WITH $\mathcal{DRC}$ AND $\mathcal{IC}$

**Definition 10 (Disjunctive Referential Constraint)** A disjunctive referential constraint ($\mathcal{DRC}$) is an expression of the form

$$\forall(Y_1, ..., Y_m) \ [r(Y_1, ..., Y_m) \rightarrow \exists(Z_1)s_1(X_1) \lor ... \lor \exists(Z_u)s_u(X_u)].$$

where $s_1, ..., s_u (1 \leq u)$, and $r$ are predicate names; $Y_1, ..., Y_m$ and $Z_1, ..., Z_u$ are different variables. $X_1, ..., X_u$ are tuples of variables and constants; for any variable $V \in X_i (1 \leq i \leq u)$, if $V \notin \{Y_1, ..., Y_m\}$, then $V \in Z_i (1 \leq i \leq u)$. Note that if $u = 1$, the constraint will be reduced to a referential constraint as described in [ZÖ97].

In first order logic, skolemization is used to eliminate existential quantifications without loss of information. For example, the skolemized form of the constraint $\mathcal{drc}$ in Example 20 is:

$$\text{empl}(X) \rightarrow \text{member}(X, p1(f_1(X))) \lor \text{member}(X, p2(f_2(X))).$$

where $f_1$ and $f_2$ are unique Skolem function symbols.

When clear from the context, the set of disjunctive referential constraints is denoted by $\mathcal{DRC}$ and the set of implication constraints is denoted by $\mathcal{IC}$. Given a database $D$, we define that if $D$ is consistent with $\mathcal{DRC}$ and $\mathcal{IC}$ if $D$ satisfies the integrity constraints. Next we give the formal definition of consistency.

**Definition 11 (Consistency)** A database instance $D$ is consistent if $D$ satisfies $\mathcal{DRC}$ and $\mathcal{IC}$ in the standard model-theoretic sense, that is, $D \models \{\mathcal{DRC}, \mathcal{IC}\}$; $D$ is inconsistent otherwise.

**Definition 12 (Containment under $\mathcal{DRC}$ and $\mathcal{IC}$)** A conjunctive query $Q$ is \{\mathcal{DRC}, \mathcal{IC}\}-contained in another conjunctive query $Q'$, denoted $Q \subseteq_{\mathcal{DRC}, \mathcal{IC}} Q'$, if $Q(D) \subseteq Q'(D)$ for any database $D$ consistent with the integrity constraints $\mathcal{DRC}, \mathcal{IC}$. $Q$ and $Q'$ are \{\mathcal{DRC}, \mathcal{IC}\}-equivalent, denoted $Q =_{\mathcal{DRC}, \mathcal{IC}} Q'$ if $Q \subseteq_{\mathcal{DRC}, \mathcal{IC}} Q'$ and $Q \supseteq_{\mathcal{DRC}, \mathcal{IC}} Q'$.

### 5.3 Query Containment with $\mathcal{DRC}$ and $\mathcal{IC}$

In [ZÖ97], a referential expansion is introduced to rewrite the original query to a unique expanded query, which reflects a respective referential constraint. However, in the case of a disjunctive referential constraint, we obtain a set of expanded queries. The technique we apply, is borrowed from the minimal model semantics for disjunctive logic programming [LMR92].
5.3.1 Disjunctive Referential Expansion

**Definition 13 (Disjunctive Referential Expansion)** Let $\mathcal{DRC}$ be a set of disjunctive referential constraints and $Q$ a conjunctive query of the form

$$q(\bar{X}) \leftarrow p_1(\bar{Y}_1), ..., p_n(\bar{Y}_n), I.$$  

Let $F$ denote the set of atoms in the body of $Q$, namely, \{\(p_1(\bar{Y}_1), ..., p_n(\bar{Y}_n)\}\}, and $I$ a conjunction of inequalities.

1. Let $M$ be any set of atoms such that for any atom $T$ of $M$, if there is a $\mathcal{DRC}$ rule in $\mathcal{DRC}$ of the form as in Definition 10 and a substitution $\rho$ from $r(Y_1, ..., Y_m)$ to $T$, then there is at least one of $\rho(s_i(\bar{X}_i))(1 \leq i \leq u)$ in $M$.
2. Let $\mathcal{M}$ be the set of all such $M$, for which in addition there holds $F \subseteq M$.
3. Let $\text{min}(\mathcal{M}) = \{M \in \mathcal{M} : \exists M' \in \mathcal{M}, M' \subset M\}$.
4. We enumerate the elements of $\text{min}(\mathcal{M})$ by \{\(F'_1, ..., F'_k\}\}.
5. We give each skolem function in $F'_i(1 \leq i \leq k)$ to a distinct variable name which does not occur in $F'_i(1 \leq i \leq k)$. Finally the set \{\(F'_1, ..., F'_k\}\} is obtained by the renaming of each element in \{\(F'_1, ..., F'_k\)\}.

The Disjunctive Referential Expansion of $Q$ using $\mathcal{DRC}$ is the set of sub-queries denoted $Q^e = (Q^e_1, Q^e_2, ..., Q^e_k)$. Each $Q^e_i$ has the form $q(\bar{X}) \leftarrow F_i, I_i$, where $F_i$ is the set of atoms as defined above.

Note that at the last step, we simply replace each skolem function with a distinct variable, so that the expanded sub-queries fall into the category of function free conjunctive queries. We argue that since such new generated variables do not appear at the head of each expanded query, using distinct variables is a natural way expressing the existential quantifiers in the $\mathcal{DRC}$. The semantical correctness is proven in [AHV95].

It should be noticed that if there are more $\mathcal{DRC}$ rules, the disjunctive referential expansion is not trivial any more. Example 21 shows one expansion.

**Example 21 (Containment of CQs under disjunctive integrity constraints)**

Let $\text{teach}(X, Y)$ be the relational schema meaning someone $X$ teaches the course $Y$, and $\text{emp}(X)$ that $X$ is a employee, and so on. There are two disjunctive referential constraints as follows:

\[
\begin{align*}
\text{drc1} : \quad & \text{teach}(X, Y) \rightarrow \text{graduate}(X) \lor \text{faculty}(X). \\
\text{drc2} : \quad & \text{emp}(X) \rightarrow \text{faculty}(X) \lor \text{staff}(X).
\end{align*}
\]

The constraint $\text{drc1}$ can be explained as: if someone teaches one course, then he must be either a graduate or a faculty. The constraint $\text{drc2}$ means that if an
employee must be either a faculty or a staff. Now the query is given of getting the people who teach a course and is also an employee of the university:

$$Q : q(X) \leftarrow \text{teach}(X,Y), \text{emp}(X).$$

If we expand the body of the query $Q$, using the constraints $\text{drc1}$ and $\text{drc2}$ above, then the final expansion set consists of two sub-queries:

$$Q^e = \{Q_1^e, Q_2^e\}, \text{ where}$$

$$Q_1^e : q(X) \leftarrow \text{teach}(X,Y), \text{emp}(X), \text{faculty}(X).$$

$$Q_2^e : q(X) \leftarrow \text{teach}(X,Y), \text{emp}(X), \text{graduate}(X), \text{staff}(X).$$

Note that since the atom $\text{faculty}(X)$ appears in both disjunctive referential constraints, there is only one sub-query that contains it, from the definition of minimal.

From the above definition and example, it can be easily seen that the general algorithm of model generation, which collects all the minimal models of a disjunctive logic programming can be used for the expansion here. The only difference is that the elements in $F$ are not ground atoms. However, this can be circumnavigated by treating all the variables in $F$ as distinct constants.

**Theorem 9 (Reduction of disjunctive expansion to minimal model semantics)**

The disjunctive referential expansion can be polynomially reduced to the problem of getting all the minimal models in disjunctive logic programming (DLP).

**Termination.** The general referential expansion procedure does not terminate [ZÖ97, AHV95]. This is the case of disjunctive referential expansion too, since the referential constraints are the special form of the disjunctive referential constraints. However, if the DRC has the acyclic property, the expansion will always terminate.

**Definition 14 (Acyclicity of referential constraints)** A set of DRC is acyclic if there is no such sequence $r_i(Y_i) \rightarrow S_{l,i}(X_{i,1,i}) \lor ... \lor S_{u,i}(X_{u,i})(i \in [1, n])$ in DRC that for $i \in [1, n], S_{l,i} = r_{i+1}$ for $i \in [1, n-1], l \in [1, u_i]$, and $S_{l,n} = r_1(l \in [1, u_n]).$

**Proposition 6 (Termination of disjunctive expansion)** Given a conjunctive query $Q$ and a set of acyclic DRC, the expansion of $Q$ using DRC terminates after an exponentially bounded number of steps.

A DRC is full constraint if it has no existential quantifiers. The disjunctive expansion using a set of full DRC terminates – the constraints need not to be acyclic. This is because the reduced form has the semantics of Disjunctive Datalog which guarantees termination [FM92, Min92].
50. CONTAINMENT OF CONJUNCTIVE QUERIES WITH DISJUNCTION

Corollary 1 (Containment of CQs under DRC) Let $Q$ be a conjunctive query and $DRC$ a set of disjunctive referential constraints, and $Q^e = (Q_1^e, Q_2^e, ..., Q_k^e)$ is the union of disjunctive referential expansion; given any database $D$ consistent with the integrity constraints $DRC$, if one tuple $t \in Q(D)$, then there is at least one $Q_i^e \in Q^e (1 \leq i \leq k)$, such that $t \in Q_i^e(D)$.

**Theorem 10 (Containment of CQs under DRC)** Given a query $Q$, a set of queries $V$ with the same format as $Q$, a set of $DRC$; $Q^e = (Q_1^e, Q_2^e, ..., Q_k^e)$ is the union of the disjunctive referential expansion of $Q$, then $Q \subseteq V$ in the presence of $DRC$ (written as $Q \subseteq_{DRC} V$), if and only if for each $Q_i^e (1 \leq i \leq k)$, there is $Q_i^e \subseteq V$.

### 5.3.2 Containment Checking Algorithm

Considering the presence of both constraints, the containment checking is processed in two steps: (i) Firstly we expand the query to an equivalent set of sub-queries using the disjunctive expansion; (ii) Secondly the containment checking under implication constraints of each sub-query is executed. It is formalized as follows:

**Theorem 11 (Containment of CQs under DRC)** Given a query $Q$, a set of queries $V$ with the same format as $Q$, a set of $DRC$ and a set of $IC$; $Q^e = (Q_1^e, Q_2^e, ..., Q_k^e)$ is the union of the disjunctive referential expansion of $Q$. $Q \subseteq_{DRC, IC} V$ if and only if for each $Q_i^e (1 \leq i \leq k)$, there is $Q_i^e \subseteq IC V$.

According to [ZÖ97], $Q_i^e \subseteq IC V$ means $Q_i^e \subseteq V$ in the presence of $IC$. To test, there must be symbol mappings from the $V$ or $IC$ to $Q_i^e$. For the case of inequalities in the query, there must be an implication test from the inequalities of $Q_i^e$ to the disjunction of that of $V$ [ZÖ97].

The next example illustrates the containment checking algorithm in the presence of both disjunctive referential and implication constraints:

**Example 22 (Containment of CQs under DRC)** Given the $drc$ and query $Q$ as in Example 20, the disjunctive expansion of $Q$ is the union of the four sub-
5.4. CONCLUSION

queries:

\[ Q_1 : \quad q(X, Y) \leftarrow \text{empl}(X), \text{empl}(Y), \text{same\_skill}(X, Y), \text{member}(X, p_1, X_1), \text{member}(Y, p_1, Y_1). \]

\[ Q_2 : \quad q(X, Y) \leftarrow \text{empl}(X), \text{empl}(Y), \text{same\_skill}(X, Y), \text{member}(X, p_1, X_1), \text{member}(Y, p_2, Y_2). \]

\[ Q_3 : \quad q(X, Y) \leftarrow \text{empl}(X), \text{empl}(Y), \text{same\_skill}(X, Y), \text{member}(X, p_1, X_1), \text{member}(Y, p_1, Y_1). \]

\[ Q_4 : \quad q(X, Y) \leftarrow \text{empl}(X), \text{empl}(Y), \text{same\_skill}(X, Y), \text{member}(X, p_2, X_2), \text{member}(Y, p_2, Y_2). \]

\[ \square \]

Note that we replace \( f_1(X) \) with \( X_1 \), \( f_2(X) \) with \( X_2 \), \( f_1(Y) \) with \( Y_1 \) and \( f_2(Y) \) with \( Y_2 \) respectively.

The containment tests will give the following results: There is a containment mapping from \( \text{ic} \) to \( Q_1 \) and \( Q_4 \); \( Q_2 \subseteq Q_1 \); \( Q_3 \subseteq Q_2 \). As a result, we get that \( Q \subseteq_{\text{DRC,IC}} Q_1 \cup Q_2 \).

Complexity

The expansion of a conjunctive query \( Q \) using a set of acyclic \( \text{DRC} \) is decidable, but intractable, since the implication of an inclusion dependency (ind) by an acyclic set of ind’s is NP-complete [AHV95]. In dealing with the full disjunctive expansion, the complexity of expansion is equivalent to that of minimal model generation of Disjunctive Datalog, which has been proved to be \( \Pi_2 \)-complete [DEGV97], however, the data complexity here is in terms of the size of the query. Usually the size of the query is very small compared to that of database. Therefore, our techniques are still of practical interest.

5.4 Conclusion

Disjunctive integrity constraints are crucial dealing with incomplete information in the database [BLR00]. Actually, it is the general form of integrity constraints introduced in [GGGM98]. Referring to the minimal model semantics of DLP, we were able to solve the query containment problem under disjunctive referential and implication constraints. As mentioned in the introduction, in dealing with semi-structured data and XML, the referential constraints in the form of disjunction have been proposed in the work of the translation of XPath to relational query model [DT01]. An extension of the \textit{chase} algorithm is given in [DT01]. However, we argue that without using the\textit{ minimal} semantics, the expanded \textit{chase} tree could have an
52 5. CONTAINMENT OF CONJUNCTIVE QUERIES WITH DISJUNCTION

exponential blow-up in the size of disjunctive referential constraints. Furthermore, the referential and implication constraints are the generalized form of inclusion and functional dependencies, so that our method can deal with the problem in [DT01], but not vice versa.
Part III

Query Containment on Tree Databases
Preliminaries on XML

6.1 Databases with Tree Structures

6.1.1 XML

The design of XML (Extended Markup Language) was driven by the idea to have a generic SGML-based language for representing semistructured data in a self-describing way, i.e., the instance contains both the contents, and a description of the semantics of the contents. The latter is achieved by structuring the contents using semantical tags: every XML application defines its own semantical tags which are described in a DTD (Document Type Description). The elements represent objects of an application area, containing information both in the attributes and in the element contents.

The abstract data model of an XML instance (we use XML instance and XML database interchangeably in this section) is a tree, consisting of nodes. The XML data model distinguishes different types of nodes, amongst them document nodes (the entry points of the trees), element nodes (the inner tree nodes), text nodes (leaves), and attribute nodes (another type of leaves). The DOM (Document Object Model) API is a specification of an abstract datatype which implements this data model.

Every XML instance is associated with a unique document node which serves for accessing the XML instance. The document node again has a unique child, which is the root node of the document.

- **Element nodes** (including the root node) have a name (often referred to as their "tag"). The element contents is an ordered list of children, i.e., element nodes and text nodes. Additionally, elements may have a unordered set of attribute nodes associated with them.

  The XML tree structure is recursive: every element node together with its children and attributes is again a tree.

- **Text nodes** contain only text contents and have no children and no attributes.
• Attribute nodes have a name (e.g., name or taught_by), a type, and a value. The value can be either atomic, or a set of atomic values. The basic XML model distinguishes between the following attribute types:
  – CDATA: scalar, arbitrary text contents,
  – NM_TOKEN: scalar, token values (i.e., text without whitespaces),
  – NM_TOKENS: multivalued, a list of tokens (separated by whitespace),
  – ID: a distinguished scalar NM_TOKEN attribute. Its value must uniquely identify an element throughout the whole XML instance.
  – IDREF: a scalar reference attribute, its value must occur as the value of an ID attribute somewhere in the XML instance.
  – IDREFS: a multivalued reference attribute, each of its values must occur as the value of an ID attribute somewhere in the XML instance.

![XML tree diagram](image)

**Figure 6.1: A XML tree**

**Tree or graph**

In spite of many bells and whistles, XML can be simply considered as a labeled ordered tree, possibly with data values associated to the leaves. As a result, we do not take features such as IDREFS, which turns XML into a graph, into consideration.
6.1. DATABASES WITH TREE STRUCTURES

Ordered or unordered

Although there are still unordered features, such as the attribute nodes, XML is claimed to be ordered.

**Definition 15 (Document Order)** Every XML tree defines an enumeration of its elements, called document order ($<_{doc}$) which results from traversing the tree recursively by depth-first search, listing the root element before traversing the sub-tree.

The notion of document order is extended to attribute nodes by $attr_1 <_{doc} attr_2$ if $elem_1 <_{doc} elem_2$ holds where $elem_i$ is the element to which $attr_i$ belongs. The order of different attribute nodes of one element is arbitrary.

6.1.2 XML data model representation

Among others, there are generally two ways representing the XML data model:

- **Native.** XML data are stored in a database system, which is based on the DOM specification and augmented with index structures. Commercial systems such as Tamino [Tam] and eXcelon [eX] use this method.

- **Shredding into RDBMS.** There are numerous research proposals on storing and querying XML data in RDBMS [TVB+02]. Commercial database products such as SQL Server, Oracle and DB2 also provide support for data manipulation of XML.

The problem of storing XML data without order is as follows: (1) queries asking for descendant edge have to be executed with the transitive closure of the relation of parent-child edge, and consequently, the complexity of both query evaluation and containment would be unnecessarily high. Another alternative, which storing all the transitive closure of parent-child relations require too much space, is obviously not a feasible solution; (2) the document order information could be lost. For instance, queries asking for the "first child of one node" require that the order on the children of this node is stored.

As a result, to capture the semantics of the document order, we have to first "encode" the XML data. Next we introduce two encoding methods, the "inverted list" method and the "Dewey order encoding".

**Inverted List Encoding**

Zhang et al. [ZND+01] proposed the "inverted list" method by assigning for each node a start position, an end position, and a nesting depth.
The inverted list encoding method works as follows: there are two indexes: E-index, storing indexed elements and T-index, storing indexed text words. Each inverted list records the occurrences of an element or a word. Each occurrence is indexed by its document number, its position and its nesting depth within the document. The example in Figure 6.2 illustrates this. Observe that for each element in the XML excerpt (Figure 6.1), there is a 3-tuple (docno, begin : end, level) associated with it, where docno is the identifier for the document, begin and end denote the position of the element node generated by counting the word numbers in the XML tree with a depth-first traversal. Since each non-leaf node is always traversed twice, the begin position records the number before all its children are visited and the end position records the number after that. The level tag denotes the nesting depth of the node within the document. Similarly for each word node, a 3-tuple (docno, wordno, level) is used to represent it. The only difference between the element node and word node is that text words are all leaf nodes, so that they are traversed only once.

Dewey Order Encoding

Dewey order is based on Dewey Decimal Classification developed for general knowledge classification. Tatarinov et al. [TVB+02] compared the different encoding
methods with respect to the update of XML data. Dewey order turned out to achieve the best performance for a mix of queries and updates. Interestingly, LDAP deploys the similar encoding method to give each node a unique ID, which is called oc.

The idea is as follows: each node is assigned a vector that represents the path from the document’s root to the node. Each component of the path represents the local order of an ancestor node. Figure 6.3 shows the Dewey order encoding of the same XML tree.

![Dewey Order encoding](image)

Figure 6.3: Dewey Order encoding

The advantage of Dewey order encoding is that parent-child relation, as well as ancestor-descendant relation, can be easily expressed with substring operation. Furthermore, queries such as "find the third child node" tends to be more efficient than the "inverted list" method.

### 6.2 The XML query model XPath

XPath [CD99] defines the basic addressing mechanism in XML documents, which is employed by most XML querying languages. The expressions defined by XPath are called location paths. Every location path declaratively selects a set of nodes
from a given XML document. For instance, the expression

```
/teachers/teacher/research/subject/
```

gathers all nodes $N$ such that $N$ is a subject element of some research element which in course is a subelement of some teacher subelement of a teachers subelement of the root element.

**Location steps**

XPath is based on navigation through the XML tree by path expressions of the form `//step/step/.../step`. Formally, the input to every location step is a node set, called the context. From this set, a new node set (called result set) is computed which then serves as input for the next step. For this computation, the input node set is processed, evaluation the location step for every node in it, appending its result set to the overall result, and proceeding with the next node. Every single step is of the form

```
axis::nodetest[filter]*.
```

which specifies that navigation goes along the given axis in the XML document. The axis specifies the tree relationship between the nodes selected by the location step and the current context node. Along the chosen axis, the nodetest specifies the node type and the name of the nodes to be selected. From these, the ones qualify which satisfy the given filter (which in turn contains predicates over XPath expressions). If more than one filter are given, they are applied iteratively.

The semantics of XPath expressions is defined in terms of node sets, i.e., unordered forests.

**Axes**

For every navigation step, the axis specifies the direction of navigation in the tree. All forward axes (denoted by (f)) enumerate the nodes in document order, whereas all backward axes (b) enumerate them in reverse document order.

**Definition 16 (XML Axes)**  Given an element, every axis defines a list of nodes:

- **self (f)**: contains exactly the element itself,
- **child (f)**: enumerates all subelements of the element,
- **descendant (f)**: enumerates all subelements by depth-first search of the element,
- **parent (f)**: contains exactly the parent of the element,
- **ancestor (b)**: enumerates the ancestors of the element, starting with the parent,
6.2. THE XML QUERY MODEL XPATH

- **following-sibling** (f): enumerates all the same level elements whose global document order is greater than that of the element, and which share the same parent as the element,

- **preceding-sibling** (b): enumerates all the same level elements whose global document order is less than that of the element, and which share the same parent as the element,

- **following** (f): enumerates all the elements whose global document order is greater than that of the element, excluding all the ancestor elements,

- **preceding** (b): enumerates all the elements whose global document order is less than that of the element, excluding all the ancestor elements,

- analogous, **descendant-or-self** (f), **ancestor-or-self** (b),

- **attribute**: enumerates all attributes of the element.

**Node tests**

The second part of a step is the **nodetest**. It specifies the node type and the name of the nodes to be selected by the location step. Every axis has a principal node type: For the attribute axis, the principal node type is attribute, for all other axes, the principal node type is element including PCDATA (text) elements. In the location steps, the node test is one of the following:

- a name, which selects all subelements of the context node having the given name,

- the test on **text** contents, which selects all PCDATA subelements of the context node,

- the test on **element** contents, which selects all element children of the context node,

- * (wildcard), selects all element nodes.

**Filters in location steps**

Using filters, the node list obtained from the nodetest is further restricted - selecting only those nodes which satisfy the given filter. When evaluating a filter, the node list selected by the above steps (navigation along an axis and applying the nodetest) is called the **context**, and the currently processed node is the **context node**. The filter contains **predicates over expressions**, i.e., terms of the following form:

- boolean over predicates.
• arithmetic expressions over number.
  - `number()` returns the numerical representation of the object specifies as an argument,
  - `sum()` returns the addition of its arguments,
  - `floor()` returns the largest integer not greater than the argument,
  - `ceiling()` returns the smallest integer greater than the argument,
  - `round()` returns the closest integer to the argument.

• string operations.
  - `string()` creates a string from a given object,
  - `concat()` returns the concatenation of its arguments,
  - `starts-with()` returns true if the first argument to the function starts with the contents of the second argument,
  - `substring-before()` returns the substring that precedes the first occurrence of the second argument,
  - `substring-after()` returns the substring that follows the first occurrence of the second argument,
  - `substring()` returns the substring between the two indices specified as arguments to the function,
  - `string-length()` returns the size of its argument,
  - `normalize-space()` returns the string resulting from normalizing the reoccurring sequences of spaces in the argument string,
  - `translate()` replaces in the first argument occurrences of the second argument with characters in the third argument.

• Nodetest functions
  - `last()` returns $n$ such that $n$ is the size of the context,
  - `position()` returns the index of the context node in the current context (i.e., "5" if the context node is the 5th node in the node list which remained after evaluating the nodetest). The filter `position()`=i may be abbreviated by `[i],
  - `count(nodeset)` returns the number of nodes in nodeset, e.g., `count(teach)` returns the number of `teach` subelements of the current context node,
  - `id(expr)` returns the node(s) in the current XML instance whose id(s) result from evaluating `expr` w.r.t. the context node.
6.3 DTD (Document Type Definition)

XML marks the "return of the schema" in semistructured data in the form of Data Type Definitions (DTDs). This is significant, because schematic information is essential at all levels of database design, implementation, and usage. The DTD describes a document type by specifying which tags are allowed, their attributes, and the allowed nestings. Roughly, the DTD corresponds to the schema definition in relational or object-oriented databases. For electronic data interchange, it is necessary that all partners agree on a common DTD, e.g., car producers and suppliers.

A DTD for a document type doctype consists a grammar which describes the class of documents,

- which elements are allowed in a document of the type doctype,
- which subelements are allowed for these elements (element types, order, cardinality),
- which attributes are allowed (attribute name, type, and cardinality),
- entities may be defined.

The structure of contents of elements is defined by element type declarations

```xml
<!ELEMENT elem_type contentmodel>
```

where

- if contentmodel = EMPTY, elements of type elem_type are empty, i.e., do not have any contents.
- if contentmodel = (#PCDATA), elements of type elem_type have only text contents and perhaps attributes,
- if contentmodel = (expression) where expression is regular expression over element names, the structure of a complex element type is described. The following constructs are used:
  - ",": sequence,
  - "|": exclusive-or,
  - "*": arbitrary often,
  - ":": arbitrary often, at least once,
  - "?:": optional (0 or 1 times).

There are much more bells and whistles in the standard definition of DTDs, which will be not described here. Essentially, a DTD is an extended context-free grammar.
The non-terminals of the grammar are the labels (tags) of elements in the labeled tree corresponding to the XML document. There are no terminal symbols. Let \( \Sigma \) be a finite alphabet of labels. A DTD consists of a set of rules of the form \( e \rightarrow r \) where \( e \in \Sigma \) and \( r \) is a regular expression over \( \Sigma \). There is one such rule for each \( e \), and the DTD also specifies the label of the root. An XML document satisfies a DTD if it is a derivation of the extended context-free grammar. That is, for label \( e \) with associated rule \( e \rightarrow r \), and each node labeled \( e \), the sequence of labels of its children spells a word in \( r \). For example, a DTD might consist of the rules:

\[
\begin{align*}
\text{root : section;}
\text{section } & \rightarrow \text{ intro, section*, conc}
\end{align*}
\]

An example of a labeled tree satisfying the above DTD is

![Diagram of an XML tree satisfying a DTD](image)

Figure 6.4: An XML satisfies the DTD

Each DTD \( d \) defines a set of labeled ordered trees, denoted by \( \text{sat}(d) \).

**Comparison of DTD and XPath with respect to expressiveness**

In spite of the different functionalities of XPath and DTD, it is interesting to have some impression on the expressive power of them on XML.

As mentioned above, DTD is an extended context-free grammar on trees. Interestingly, there is an extension of DTD, called specialized DTD [PV00], which decouples the names and types from label names. It is shown that specialized DTD is precisely equivalent to tree automata on both ranked and unranked trees [PV00]. Recently, Marx [Mar04] proposed the results on the expressive power of XPath by pointing out that an extension of XPath, called XPath, with conditional axis feature added, is exactly equivalent to first-order logic.

It is obvious that in some sense, DTD is much stronger than XPath, if the expressiveness is considered. However, regular expressions in DTD can be too powerful
for most practical situations. For instance, they allow stating properties unlikely to be useful, such as "the number of sections must be even" [Via01]. More restricted formalisms, such as star-free regular languages (definition cf. Chapter 7), are often sufficient. It is well-known that star-free regular languages are equivalent to first-order logic on finite words. It turns out that replacing regular languages with star-free languages decreases the complexity of various problems related to typing [AMN⁺03].
QUERY EQUVALENCE OVER WORDS

Finite words can be considered as a special fragment of finite ordered trees. The knowledge of query languages applied to finite words is crucial for the further understanding of ordered trees. In this chapter, we consider queries over finite words.

We first give definitions of regular languages and star-free regular languages, which is followed by the mechanism for encoding the finite words as linearly ordered databases. On the basis of this structure, we propose a polynomial time algorithm for the containment checking of a fragment of XPath: $XP(\//,\//,\ast)$ over finite words.

7.1 Regular Expressions and Star-free Expressions

7.1.1 Regular Expressions and Languages

We consider a finite alphabet $A$. $A^*$ represents the set of all finite words on the alphabet $A$. A language $L$ over the alphabet $A$ is a subset of $A^*$. Constants $\emptyset$ and $\epsilon$ denote empty set and empty word respectively.

**Definition 17 (Regular expression)** Over the alphabet $A$, $E$ is a regular expression if $E$ is

1. $a$, where $a \in A$,
2. $\epsilon$,
3. $\emptyset$,
4. $(E_1 \cup E_2)$, where $E_1$ and $E_2$ are regular expressions,
5. $(E_1 \cdot E_2)$, where $E_1$ and $E_2$ are regular expressions,
6. $(E_1^*)$, where $E_1$ is a regular expression,
7. $(E_1 \cap E_2)$, where $E_1$ and $E_2$ are regular expressions,
8. $(\sim E_1)$, where $E_1$ is a regular expression.

It is shown that the same expressive power can be achieved without complementation $(\sim)$ and intersection $(\cap)$. The expressions are also referred as extended regular
expressions. However, adding them will help us in the exposition of star-free expressions in the later section.

**Definition 18 (Regular language)** Given a regular expression $E$, the language of the regular expression $L(E)$ is defined as:

1. $L(a) = \{a\}$.
2. $L(\emptyset) = \emptyset$.
3. $L(\epsilon) = \{\epsilon\}$.
4. $L(E_1 \cup E_2) = L(E_1) \cup L(E_2)$.
5. $L(E_1 \cdot E_2) = \{uv|u \in L(E_1), v \in L(E_2)\}$.
6. $L(E_1^*)$ is the smallest set containing $\epsilon$ such that $v \in L(E_1)$ and $w \in L(E_1^*)$ implies $vw \in L(E_1^*)$.
7. $L(E_1 \cap E_2) = L(E_1) \cap L(E_2)$.
8. $L(\sim E_1) = A \cdot \{} L(E_1) \}$. □

The equivalence and containment problem of regular expressions is similarly defined as query equivalence and containment problem for database queries. Given two regular expressions $E_1$ and $E_2$, $E_1$ is contained in $E_2$, denoted as $E_1 \subseteq E_2$ if and only if the language $L(E_1)$ is a subset of the language $L(E_2)$. $E_1$ is equivalent to $E_2$ if they define the same language.

It is well-known that the equivalence problem for DFA is polynomial time ($n^2$). However, the equivalence problem for regular expressions and NFA is NP hard [Sch99]. In fact, the equivalence problems of both NFA and regular expressions are Co-NP complete [GJ78]. This is because converting from NFA to DFA and from regular expression to DFA increases the size of the states exponentially. This involves a property of so-called succinctness, which is recently studied by Grohe and Schweikardt [GS04]. Two formalisms may have same expressive power, but one may be much more succinct than the other. As a result, the satisfiability and model checking complexity results could differ significantly.

### 7.1.2 Star-free Regular Languages

**Definition 19 (Star-free regular expression)** A regular expression $E$ is star-free, if it does not contain any occurrence of kleene star $*$ operator. □

Accordingly, star-free regular expressions over a given alphabet $A$ are built up from constants $\emptyset$, $\epsilon$ and $a \in A$ by means of the operations $\cup, \cap, \sim$, and concatenation dot $\cdot$. Note that the expression $A^*$ is admitted, as abbreviation of $\sim \emptyset$. 
7.2. WORD MODELS

It should be emphasized that many seemingly non-star-free languages are in fact star-free. For instance, given the alphabet \( A = \{a, b, c\} \), the language defined from the expression

\[
ab * c
\]

is star-free. Because the expression can be rewritten as follows:

\[
E_2 = a(\sim E_1)c
\]

where \( E_1 \) is as follows:

\[
E_1 = (A * aA*) \cup (A * cA*)
\]

Intuitively, \( E_1 \) defines the language which contains either \( a \) or \( c \), and the negated form of \( E_1 \) defines the language that only contains \( b \).

The equivalence problem of star-free regular expressions is Co-NP complete, because the co-problem, the so-called Star-Free Regular Expression Inequivalence problem is NP-complete (p. 76 [GJ78]).

7.2 Word Models

In order to apply the queries on finite words, we have to treat words as relational structures. Considered as databases, words can be modeled as follows [Tho90]. Let \( A \) be a finite alphabet and let \( w = a_1, \ldots, a_n \) be a word over \( A \). The word \( w \) is represented by the relational structure

\[
w = (\text{dom}(w), S, <, (P_a)_{a \in A})
\]

where \( \text{dom}(w) = \{1, \ldots, n\} \) is the set of positions of \( w \), \( S \) is the successor relation on \( \text{dom}(w) \), with \((i, i+1) \in S \) for \( 1 \leq i \leq n \), \( < \) is the natural order on \( \text{dom}(w) \), and \( P_a \) is a unary predicate and corresponds to the position in \( w \) carrying an \( a \).

For instance, let \( A = \{a, b\} \) and \( w = abbab \), the word model (or database instance) is as follows:

\[
D = \{S(1, 2), S(2, 3), S(3, 4), S(4, 5), P_a(1), P_a(4), P_b(2), P_b(3), P_b(5)\}
\]
7.3 Equivalence of logics on finite ordered structures

7.3.1 Star-free regular languages and FO[<]

**Definition 20 (FO[<])** The logic FO[<] is built using variables \( x, y, \ldots \) ranging over positions in word models, and is composed of the following atomic formulas

\[
x = y, \quad S(x, y), \quad x < y, \quad P_a(x)_{a \in A}
\]

by means of the connectives \( \neg, \land, \lor \) and the quantifiers \( \exists \) and \( \forall \).

Similarly, by FO\([S]\) we mean the first order logic without use of \( < \).

A language \( L \subseteq A^* \) is FO[<]-definable if a first-order sentence \( \varphi \) with \( L = L(\varphi) \).

One of the main results stating the relationship between the FO[<] and star-free regular language is the following theorem from McNaughton and Papert. Note that there is another equivalence between star-free regular language and counter-free finite state automaton, which will not be explained in detail.

**Theorem 12 (Equivalence of star-free regular language and FO)** [RS71]

A language \( L \) is star-free if and only if there exists a FO[<] sentence \( \varphi \) such that \( L = L(\varphi) \), and vice versa.

**Example 23 (Star-free languages to first-order sentences)** Over the alphabet \( A = \{a, b, c\} \), the star-free regular language \( L_1 \), defined with the expression:

\[
a \cdot b \ast c \cdot A^*
\]

can be defined with the following sentence:

\[
P_a(1) \land \exists z(1 < z \land P_c(z) \land \forall y(1 < y < z \rightarrow P_b(y))).
\]

Over the same alphabet \( A \), the star-free regular language \( L_2 \), defined with the expression as follows:

\[
A \ast \cdot a \cdot b \cdot \sim (A \ast \cdot a \cdot A^*)
\]

can be defined with the following sentence:

\[
\exists x \forall y(S(x, y) \land P_a(x) \land P_b(y) \land \exists z(y < z \land P_a(z))).
\]
The sub-expression \((A^* \cdot a \cdot A^*)\) represents a word with \(a\) occurs, and its complement \(\sim (A^* \cdot a \cdot A^*)\) defines the language that does not contain \(a\), which implies that only \(b\) or \(c\) could occur, given the alphabet \(\{a, b, c\}\).

The regular expression \((aa)^*\), on the other hand, is not star-free, since it expresses the even length of the string, which is not expressible in \(\text{FO}[\prec]\).

**Theorem 13 (Equivalence of PLTL and FO)** [Kap68] A finite language is PLTL definable if and only if it is \(\text{FO}[\prec]\) definable.

### 7.4 XPath Queries over Words

In this section, we concentrate on an XPath fragment, denoted as \(\text{XP}(/, //, *)\), over finite words. The axes related to branching, such as \text{following-sibling} and \text{preceding-sibling}, as well as \text{pred}, will not be considered.

#### 7.4.1 \(\text{XP}(/, //, *)\) on Words

We consider the XPath fragment which is composed of \text{child} and \text{descendant} axes and the wildcard "." To simplify the presentation, we use the abbreviated syntax ‘/’ for \text{child} and ‘//’ for \text{descendant} respectively. We refer this fragment as \(\text{XP}(/, //, *)\). Note that the term \(\text{XP}(L)\) will be used in later sections representing the XPath fragment where \(L\) is a list of the allowed components in abbreviated notation.

#### 7.4.2 \(\text{XP}(/, //, *)\) queries and star-free regular expressions

Clearly, \text{descendant} axis is the transitive closure of \text{child} axis, which implies – if the underlying structure is unordered – that recursion is involved. However, we show in the next proposition that any \(\text{XP}(/, //, *)\) query can be expressed with a star-free regular expression, and correspondingly with \(\text{FO}[\prec]\).

**Proposition 7 (Transformation of \(\text{XP}(/, //, *)\) to star-free regular expression)**

For each \(\text{XP}(/, //, *)\) query over finite words, there is an equivalent star-free regular expression.

Given a finite alphabet \(A\) (note that we treat the tag names as single symbols, which is not always the case in the real XML data model. However, since the tag names are finite, this assumption does not affect the analysis of expressiveness and the containment checking algorithm), we have the following simple translations:
• tag (symbol constant)
  keeps unchanged.
• * (wildcard)
  can be simply translated into $A$.
• / (child axis)
  given the expression $X/Y$, where $X$ and $Y$ are either tag name or wildcard, the corresponding regular expression is $X \cdot Y$.
• // (descendant axis)
  given the expression $X//Y$, where $X$ and $Y$ are either tag name or wildcard, the corresponding regular expression is $X \cdot A \cdot Y$.

Figure 7.1 illustrates the star-free regular expression translations of four typical XP(/,//,*) queries.

$$
\begin{array}{cccc}
| & | & | & |\\
\toprule
a & a & a & a \\
\mid & * & * & * \\
\mid & b & b & b \\
\bottomrule
\end{array}
$$

$$
\begin{array}{cccc}
| & | & | & |\\
\toprule
a \cdot A \cdot b & a \cdot A \cdot A \cdot b & a \cdot A \cdot A \cdot b & a \cdot A \cdot A \cdot A \cdot b \\
\bottomrule
\end{array}
$$

Figure 7.1: XP(/,//,*) queries and the corresponding star-free expressions

Comparison of expressive power

Over finite words, the XP(/,//,*) fragment does not have the same expressive power as star-free regular expressions. For instance, the expression $a \cdot b \cdot c$ is not expressible with XP(/,//,*) , although it is star-free. In fact, the translated regular expression of an XP(/,//,*) query is simply the concatenation of symbol constants, $A\cdot s$ and $A\cdot s$, without negation ($\sim$) and union ($\cup$) involved.

7.4.3 Containment

As mentioned before, the containment problem of star-free regular expressions is Co-NP complete. However, better bound can be achieved for the containment problem of XP(/,//,*) queries. In the following, we show that there is a polynomial time containment checking algorithm for the fragment XP(/,//,*).
7.4. XPATH QUERIES OVER WORDS

From now on, we use $XP(//,//,\ast)$ expressions to denote the translated star-free regular expressions of the $XP(//,//,\ast)$ queries.

Normal Form

Since the $XP(//,//,\ast)$ expressions are concatenations of symbol constants, $A\ast$s and $A$s, a containment mapping algorithm similar to that of CQs can be applied. However, the expressions should have normal form before the containment checking algorithm can be applied.

The normal form can be obtained from the following rewriting rules:

$A \cdot A \cdot \ldots \cdot A \rightarrow A^n$

$A^+ \rightarrow A^{0+}$

$A^{n+} \cdot A^m \rightarrow A^{(n+m)+}$  \( n \geq 0, \, m \geq 1 \)

$A^m \cdot A^{n+} \rightarrow A^{(n+m)+}$  \( n \geq 0, \, m \geq 1 \)

$A^{m+} \cdot A^{n+} \rightarrow A^{(m+n)+}$  \( m, \, n \geq 0 \)

The intuition behind the normal form is as follows. Consider the regular expressions $E_1$ and $E_2$:

$E_1 : a \cdot A \ast \cdot A \cdot b$

$E_2 : a \cdot A \cdot A \ast \cdot b$

To test whether $E_1 \subseteq E_2$, a containment mapping from $E_2$ to $E_1$ has to be given. Unfortunately, for $A$ and $A\ast$ in $E_2$, there are no corresponding mappings in $E_1$.

However, it is obvious that $E_1$ and $E_2$ are equivalent. Because they both define the language starting with $a$, ending with $b$, and with at least one letter inbetween. By rewriting the expressions into normal form, the concatenations of $A\ast$s and $A$s in the expressions are compressed so that the containment mapping can be correctly applied.

**Definition 21 (Normal Form of $XP(//,//,\ast)$ Expressions)** The normal form of $XP(//,//,\ast)$ expressions is as follows:

$$
\begin{align*}
X_{1,1} \cdot X_{1,2} \cdot \ldots \cdot X_{1,m_1} \cdot A^{r_1} \cdot \ldots \cdot A^{r_{n-1}} \cdot X_{n,1} \cdot X_{n,2} \cdot \ldots \cdot X_{n,m_n}
\end{align*}
$$

where $X_{i,j} \in A(1 \leq i \leq n, \, 1 \leq j \leq m_n)$ and $r_i(1 \leq i \leq n-1)$ is either a positive integer $l$, where $l > 0$, or $k+$ where $k \geq 0$. \( \square \)
An XP(/,/,*) expression with normal form can be considered as a concatenation of partitions $p_1, p_2, \ldots, p_n$, where $p_{2i+1} (0 \leq i \leq n/2)$ are concatenations of symbol constants (denoted as c-partition), and $p_{2i} (1 \leq i \leq n/2)$ are $A^{m+}_i$s, where $0 \leq m_i$ (denoted as $A^+_i$-partition) or $A^0_i$s, where $1 \leq m_i$ (denoted as $A^-_i$-partition). We assume that the left-most partition is a c-partition, as general XP(/,/,*) queries require.

For instance, the expression

$$a \cdot A^{3+} \cdot b \cdot b \cdot c \cdot A^{0+} \cdot d$$

is composed of c-partitions $a$, $b \cdot b \cdot c$, and $d$, which are separated with $A^+_i$-partitions $A^{3+}$ and $A^{0+}$.

The expressions in Figure 7.1 have the following normal forms:

- $a \cdot A \cdot b$ is already in normal form.
- $a \cdot A \cdot A \cdot A \cdot b$ has the normal form $a \cdot A^{1+} \cdot b$
- $a \cdot A \cdot A \cdot A \cdot b$ has the normal form $a \cdot A^{1+} \cdot b$
- $a \cdot A \cdot A \cdot A \cdot A \cdot A \cdot b$ has the normal form $a \cdot A^{1+} \cdot b$

**Theorem 14 (Normal Form of XP(/,/,*))** The rewriting of any XP(/,/,*) expression $E$ into normal form takes $O(n)$ where $n$ is the length of $E$. \hfill \Box

With the XP(/,/,*) expressions in normal form, the containment of queries can be considered as the mappings of the corresponding partitions. However, different from the conjunctive queries, the rules of containment mapping on the regular expressions need to take the transitive closure (e.g. $A^n_i$s) into account.

**Definition 22 (Containment mapping of regular expressions)** The containment mappings are given for the following partitions:

- **c-partition**
  has only the mapping with the identical expression

- **A-partition with the form $A^n$ $n \geq 1$**
  has the mapping to the expression

$$X_1^{r_1} \cdot X_2^{r_2} \cdots \cdot X_m^{r_m}$$

where $X_i (1 \leq i \leq m)$ is either $A$ or a symbol constant $a \in A$, and $r_1 + r_2 + \ldots + r_m = n$.

- **$A^+_i$-partition with the form $A^{n+}_i$ $n \geq 0$**
  has the mapping to the expression

$$X_1^{r_1} \cdot X_2^{r_2} \cdots \cdot X_m^{r_m}$$
where $X_i (1 \leq i \leq m)$ is either $A$ or a symbol constant $a \in A$, $r_i (1 \leq i \leq m)$ has the form of $k$ or $k^+$, where $k$ is an integer. Furthermore, $|r_1| + |r_2| + \ldots + |r_m| \geq n$, where

$$|r_i| = \begin{cases} 
  k & \text{if } r_i = k \\
  k & \text{if } r_i = k^+ 
\end{cases} \Box$$

Intuitively, the A-partition with the form $A^n$ has the mapping to any expression with the length of $n$, and the A+-partition with the form $A^{n^+}$ has the mapping to any expression with a minimal length of $n$. The partition $A^{0^+}$, which is the normal form of $A^*$, maps to any expression with arbitrary length.

Given an XP(/, //, *) expression $E$ in normal form, the next question of importance is, given a word $w$, whether $w \in L(E)$. For general regular expression this problem could take exponential time in the size of the expression. However, due to the simplicity of the expression, we show that the membership testing can be implemented in the similar manner as the query evaluation of conjunctive queries.

**Lemma 6** Given an XP(/, //, *) expression $E = p_1 \cdot p_2 \cdot \ldots \cdot p_n$ in normal form, and a word $w$. $w \in L(E)$ if and only if there is a containment mapping from each $p_i (1 \leq i \leq n)$ to a word $w_i$, such that $w = w_1 \cdot w_2 \cdot \ldots \cdot w_n$.

**Proof 7** • "If": there is a containment mapping from each $p_i$ to $w_i (1 \leq i \leq n)$.

We have to prove that $w \in L(E)$. Given a partition $p_i (1 \leq i \leq n)$,

- if $i$ is an odd number, then $p_i$ is a c-partition. From the containment rules in Definition 22, only the identical word can be mapped from it. As a result, for each $i$, $w_i \in L(p_i) = p_i$.

- if $i$ is an even number, and $p_i$ is an A-partition with the form $A^m$. Again according the mapping rules in Definition 22, only the word with length $m$ can be mapped. Obviously, $w_i \in L(p_i) = L(A^m)$.

- if $i$ is an even number, and $p_i$ is an A+-partition with the form $A^{m^+}$, then only the word whose length is greater than or equal to $m$ can be mapped. Obviously, $w_i \in L(p_i) = L(A^{m^+})$.

As a result, for all $i (1 \leq i \leq n)$, we obtain that $w_i \in L(p_i)$. It follows that $w \in L(E)$.

• "Only if": if a word $w \in L(E)$, we have to find a containment mapping from $E$ to $w$. Since $E = p_1 \cdot p_2 \cdot \ldots \cdot p_n$, we obtain

$$L(E) = L(p_1) \cdot L(p_2) \cdot \ldots \cdot L(p_{n-1}) \cdot L(p_n) = p_1 \cdot L(p_2) \cdot \ldots \cdot L(p_{n-1}) \cdot p_n$$
Since \( w \in L(E) \), \( w \) has the form of \( p_1 \cdot L(p_2) \cdot \ldots \cdot L(p_{n-1}) \cdot p_n \). Let \( w = p_1 \cdot w_2 \cdot p_3 \cdot w_4 \cdot \ldots \cdot p_n \), where \( w_{2i} \in L(p_{2i})(1 \leq i \leq n/2) \).

If \( p_{2i}(1 \leq i \leq (n-1)/2) \) is an \( A \)-partition \( A^m \), \( L(p_{2i}) \) is a language with length \( m \). If \( p_{2i}(1 \leq i \leq (n-1)/2) \) is an \( A^+ \)-partition \( A^{m+} \), \( L(p_{2i}) \) is a language with minimum length \( m \). According the containment mapping rules in Definition 22, for every word \( w_{2i} \in L(p_{2i}) \), there is a containment mapping from \( p_{2i} \) to \( w_{2i} \).

As a result, there is a containment mapping from \( E \) to \( w \).

\[ \square \]

**Lemma 7** Given an \( XP(//,\ast) \) expression \( E \) and its normal form \( E^n \). \( E \equiv E^n \) holds.

\[ \square \]

**Proof 8** The \( XP(//,\ast) \) expression \( E \) differs from its normal form \( E^n \) only on the representation of the concatenation of \( A \)s or \( A^+ \)s. As a result, we only have to prove that the expression of concatenation of \( A \)s and \( A^+ \)s is equivalent to its normal form.

Assume the expression \( E' \) is the concatenation of \( A \)s and \( A^+ \)s which has the following form:

\[ X \cdot X \ldots \cdot X \]

where \( X \) is either \( A \) or \( A^+ \). Assume there are \( p \) \( A \)s and \( q \) \( A^+ \)s where \( p, q \geq 1 \). Since \( A \) commutes with \( A^+ \), we can rewrite the expression as follows:

\[ \overbrace{A \cdot A \cdot \ldots \cdot A}^{p} \cdot \overbrace{A \cdot A^+ \cdot \ldots \cdot A^+}^{q} \]

According to the rule \( A^+ \cdot A^+ = A^+ \), the expression is equivalent to the following:

\[ \overbrace{A \cdot A \cdot \ldots \cdot A}^{p} \cdot A^+ \]

As a result, the language \( L(E') \) is:

\[ \overbrace{A \cdot A \cdot \ldots \cdot A}^{p} + \overbrace{A \cdot A \cdot \ldots \cdot A}^{p+1} + \ldots \]

Clearly, the language \( L(E') \) is all words with minimal length \( p \).

Now according to the normal form rewriting rules we have introduced, the normal form of \( E' \) — we call it \( (E'^n) \) — is \( A^{p+} \). From Lemma 6 it is proven that any word \( w \in (E'^n) \) iff there is a containment mapping from \( (E'^n) \) to \( w \). From the containment mapping rules we have proposed, it is obvious that \( E' \) and \( (E'^n) \) define exactly the same language.
7.4. XPATH QUERIES OVER WORDS

The polynomial time algorithm

Now we are ready to give the main theorem for the containment checking of \( \text{XP}(//\ast) \) queries.

**Theorem 15 (Containment of XP(//\ast) expressions)** Given \( \text{XP}(//\ast) \) expressions \( E_1 \) and \( E_2 \) in normal form. \( E_1 \) has \( n \) partitions \( q_1, \ldots, q_m \), and \( E_2 \) has \( n \) partitions \( p_1, \ldots, p_n \). \( E_1 \) is contained in \( E_2 \), written as \( E_1 \subseteq E_2 \), if and only if there is a containment mapping from each \( p_i (1 \leq i \leq n) \) in \( E_2 \) to a partition \( b'_i \) in \( E_1 \), where \( b'_i \) is not necessarily any \( q_j (1 \leq j \leq m) \), such that \( q_1 \cdot q_2 \cdot \ldots \cdot q_m = b'_1 \cdot b'_2 \cdot \ldots \cdot b'_n \).

One has to note that the containment mapping from \( E_2 \) to \( E_1 \) is not simply the mappings of partitions in normal form. It is possible that one c-partition in \( E_2 \) can only map a part of a c-partition in \( E_1 \), while the containment of the whole expression \( E_2 \) to \( E_1 \) still holds. Before an illustrative example is proposed, we introduce the following notations. Let \( q_i \) be any c-partition of \( E_1 \). We denote \( |q_i| \) to be the length of \( q_i \). For instance, if \( q_i = a \cdot b \cdot c \), then \( |q_i| \) is 3. Let \( l = |q_i| \), we denote \( q_i^{u,v} (1 \leq u \leq v \leq l) \) as a sub-partition of \( q_i \), which starts from position \( u \) and ends at position \( v \). For instance, \( q_i^{2,3} \) from the above \( q_i \) is the partition \( b \cdot c \). Let further \( q_j(i < j) \) be another c-partition of \( E_1 \) which is at the right side of \( q_i \), and \( q_j^{w,x} \) be a sub-partition of \( q_j \), the partition between \( q_i^{u,v} \) and \( q_j^{w,x} \), denoted as \( q_j^{w,x} - q_i^{u,v} \), is in fact \( q_i^{v+1,l} \cdot q_{i+1} \cdot \ldots \cdot q_j^{w-1} \).

**Example 24** Consider the following two expressions \( E_1 \) and \( E_2 \):

\[
E_1 : \quad \begin{array}{c}
q_1 \quad a \quad q_2 \quad a \quad A^{2+} \quad a \quad q_3 \\
p_1 \quad a \quad A^{1+} \quad p_2 \\
p_3 \quad a 
\end{array}
\]

\[
E_2 : \quad \begin{array}{c}
q_1 \quad a \quad q_2 \quad a \\
p_1 \quad a \\
p_2 \quad a \\
p_3 \quad a
\end{array}
\]

Clearly, for \( p_1 \) the only possible mapping is the first \( a \) in \( E_1 \) – we call it \( q_1^{1,1} \). This holds also for the partition \( p_3 \), which is mapped to \( q_3^{2,2} \). However, since \( p_2 \) has the containment mapping to the partition between \( q_1^{1,1} \) and \( q_3^{2,2} \), namely \( a \cdot A^{2+} \cdot a \), there is a containment mapping from \( E_2 \) to \( E_1 \).

However, it is still not obvious that the containment checking algorithm is in polynomial time. A primitive algorithm would guess for each partition \( p_i \), a corresponding partition \( b'_i \) in \( E_1 \), and then check whether \( b'_1, \ldots, b'_n \) is equal to \( E_1 \). Although the single checking of the partition takes polynomial time, the number of the combinations is exponential. In fact, one single partition \( A^* \) would be enough, since any sub-expression can be mapped from it.
To lower the complexity to polynomial time, we have to avoid the exponential explosion. Actually this is also known as the problem of the exponential explosion of \textit{join} operations for general relational queries. One solution to this problem is that if the query is acyclic, the exponential explosion can be avoided. Surprisingly, the similar strategy can be applied here perfectly. Because the partitions in the expression are connected with each other in the form of a "chain", which is the simplest form of acyclicity. As a result, we can always take two adjacent partitions and make the containment checking. As soon as the checking is finished, one partition will be discarded. In this way, the \textit{join} operation is always restricted within polynomial time.

In the following we give a polynomial time algorithm for containment checking of $E_1 \subseteq E_2$. Given $E_1$ and $E_2$ with the following normal form:

\[ E_1 : q_1 \cdot q_2 \cdot \ldots \cdot q_m \]
\[ E_2 : p_1 \cdot p_2 \cdot \ldots \cdot q_n \]

where $m$ and $n$ are odd natural numbers. $p_{2i+1}(0 \leq i \leq (n-1)/2)$ and $q_{2j+1}(0 \leq j \leq (m-1)/2)$ are c-partitions. Let each $S_{2i+1}(0 \leq i \leq (n-1)/2)$ be a set of partitions in $E_1$, which can be mapped from the partition $p_{2i+1}$ in $E_2$. $|S_i|$ is the number of elements in $S_i$. Other notations used in the algorithm are introduced before Example 24 is proposed.
7.4. XPATH QUERIES OVER WORDS

Algorithm Containment Checking\((E_1, E_2)\)

Inputs: \(E_1\) and \(E_2\) in normal form.

Begin

Initialize \(S_1, S_2, \ldots, S_n\) with empty sets;
If \(|q_1| < |p_1|\) or \(q_1^{1,p_1} \neq p_1\), return No;
Add \(q_1^{1,p_1}\) into \(S_1\);

For \(i = 1\) to \(n/2\) do

\(l = |p_{2i+1}|;\)

For \(j = 0\) to \(m/2\) do

For \(k = 1\) to \(|q_{2j+1}|\) do

If there is a containment mapping from \(p_{2i+1}\) to \(q_{2j+1}^{k,k+l-1}\)

Then

\(l_2 = |S_{2i-1}|;\)

For \(h = 1\) to \(l_2\) do

\(q = S_{2i-1}[h];\)

If \(p_{2i}\) maps \(q_{2j+1}^{k,k+l-1} - q\)

add \(q_{2j+1}^{k,k+l-1}\) into \(S_{2i+1}\);

EndIf

EndFor

EndFor

EndFor

EndFor

If \(S_n\) is empty return No;
return Yes;

End

Theorem 16 The algorithm Containment Checking determines correctly whether \(E_1 \sqsubseteq E_2\). \[\square\]

Proof 9 We use induction on the number of partitions in \(E_2\) processed, with the following induction hypothesis: after the \(c\)-partition \(p_{2i+1}(2i+1 \leq n)\) is processed, the partition \(q \in S_{2i+1}\) if and only if there is a partial mapping \(\phi\) from \(p_1 \cdot p_2 \cdot \ldots \cdot p_{2i+1}\) to \(E_1\), such that \(\phi(p_{2i+1}) = q\). Thus, when the last partition \(p_n\) has been processed, \(S_n\) is not empty if and only if there is a containment mapping \(\mu\) from \(E_2\) to \(E_1\).

- "only if":

1. Base: the induction hypothesis holds for \(p_1\), since it is initialized at the beginning of the algorithm.
2. **Induction:** we assume that c-partition $p_{2i+1}$ is processed, and let $q \in S_{2i+1}$. Following the algorithm, there should be a containment mapping from $p_{2i+1}$ to $q$, and there should be at least one partition $q' \in S_{2i-1}$, such that $p_{2i}$ is mapped to $q''$, where $q''$ is the partition between $q'$ and $q$. Let the mapping from $p_{2i+1}$ to $q$ be $\rho$ and the mapping from $p_{2i}$ to $q''$ be $\varphi$.

Since $q' \in S_{2i-1}$, by the induction hypothesis, there is a partial mapping $\theta$, from $p_1 \cdots p_{2i} \cdots p_{2i+1}$ to $E_1$ such that $\theta(p_{2i-1}) = q'$.

Clearly $\theta$, $\varphi$ and $\rho$ are consistent. Let $\phi$ be the union of $\theta$, $\varphi$ and $\rho$. we obtain that $\phi$ is the containment mapping from $p_1 \cdot p_2 \cdots \cdot p_{2i+1}$ to $E_1$, such that $\phi(p_{2i+1}) = q$.

- "if": Conversely, let $\phi$ be a partial mapping $p_1 \cdot p_2 \cdots \cdot p_{2i+1}$ to $E_1$, such that $\phi(p_{2i+1}) = q$. We have to prove that after the processing of $p_{2i+1}$, $q \in S_{2i+1}$.

Let $\varphi$, $\rho$ be the subsets of $\phi$ whose domain are $p_2i$ and $p_{2i+1}$ respectively. Let $\theta$ be the subset of $\phi$, where $\varphi$ and $\rho$ are projected out. Clearly, $\theta$ is a partial mapping from $p_1 \cdot p_2 \cdots \cdot p_{2i-1}$ to $E_1$.

Let $q' = \theta(p_{2i-1})$. By the induction hypothesis, $q' \in S_{2i-1}$. Now by the algorithm, $q$ will be obtained by the complete search in $E_1$ for all containment mappings from $p_{2i+1}$. Further, since $q' \in S_{2i-1}$ and there is also a mapping from $p_{2i}$ to the partition between $q'$ and $q$ – which is $\varphi$ we mentioned above –, $q$ will be added into $S_{2i+1}$.

The completeness can also be proven by contradiction. Assume that there is a containment mapping $\phi$ from $E_2$ to $E_1$, while there is some mapping $\rho$ where $\rho \in \phi$ from $p_{2i+1}(0 \leq i \leq (n-1)/2)$, that does not exist in $M_{2i+1}$, however the whole containment holds. According to the algorithm, that $\rho$ is not added into $M_{2i+1}$ can only happen, if there does not exist one $\varphi \in M_{2i-1}$, s. t. the subexpression $E_2 = p_{2i-1} \cdot p_{2i} \cdot p_{2i+1}$ has a containment mapping in $E_1$. Since $E_2$ is contained in $E_2$, $E_2$ does not have any containment mapping in $E_1$ either. This results in a contradiction. 

\[\square\]

**Example 25 (Containment checking)** Given two XP(/,/,*) expressions $E_1$ and $E_2$ as follows:

\[E_1 : \overrightarrow{q_1} \cdot a \cdot A^{1+} \cdot \overrightarrow{q_2} \cdot a \cdot A^3 \cdot \overrightarrow{q_3} \cdot a \cdot A^{0+} \cdot \overrightarrow{q_4} \cdot \overrightarrow{q_5} \cdot \overrightarrow{q_6} \cdot \overrightarrow{q_7}\]

\[E_2 : \overrightarrow{p_1} \cdot a \cdot a \cdot A^{3+} \cdot a \cdot A^{4+} \cdot \overrightarrow{p_2} \cdot a \cdot A^{0+} \cdot \overrightarrow{p_3} \cdot a \cdot A^{1+} \cdot \overrightarrow{p_4} \cdot \overrightarrow{p_5} \cdot \overrightarrow{p_6} \cdot \overrightarrow{p_7}\]

the containment checking algorithm is shown in Figure 7.2.
7.4. XPATH QUERIES OVER WORDS

Step 1 Firstly, we take partition \( p_1 \). Since it is on the left-most side of the expression, the only mapping is left-most partition in \( E_1: q_{11}^{1,2} \), which is \( q_1 \).

Step 2 Secondly, the partition \( p_3 \) is considered. As shown in Figure 7.2, we scan from left to right on \( E_1. q_{11}^{1,1} \) and \( q_{12}^{2,2} \) can obviously not be added into \( S_3 \) because no mapping from \( p_2 \) can be found. \( q_{31}^{1,1} \) can not be added into \( S_3 \) either, since the partition between \( q_1 \) and \( q_{31}^{1,1} \), which is \( A^{1+} \), can not be mapped from \( p_2 \) in \( E_2 \). The same holds also for the \( q_{32}^{2,2} \). As a result, there are three qualified partitions that can be mapped from \( p_3 \). Namely \( q_{33}^{3,3}, q_{51}^{5,1} \) and \( q_{71}^{7,1} \).

Step 3 Thirdly, the partition \( p_5 \) is considered. The only possible mapping partition in \( E_1 \) is the right-most partition. Now we have to check whether there is a mapping for \( p_4 \). It is obvious that the only partition is \( A^{3 \cdot \cdot 0^+} \). As a result, we have found one containment mapping as illustrated. It should be noted that in this step, the partition in \( p_1 \) is not any more considered. As a result, one considers only the partial mappings from \( E_2 \) to \( E_1 \), so that the algorithm is in polynomial time.

\[ \square \]

Theorem 17 (Complexity) Given \( XP(/,//,*) \) expressions \( E_1 \) and \( E_2 \), the containment checking of \( E_1 \subseteq E_2 \) takes polynomial time in the size of \( E_1 \) and \( E_2 \). \( \square \)
Figure 7.2: Containment Checking of Two XP(/,//,* ) Queries
8 QUERY EQUIVALENCE OVER TREES

8.1 Introduction

As XML is becoming the standard format for data exchange on the internet, there have been many proposals on storing XML documents in relational databases and translating XML queries into the corresponding relational queries. In general, XML data can be stored either as unordered or ordered. The unordered method is adapted from the semi-structured data model and object-oriented databases. With this method, each node is given a unique ID and the parent-child relation between nodes is maintained explicitly. Since parent-child and ancestor-descendant relationships are the core operation in XML query processing, ancestor-descendant relation has to be obtained from recursively applying the join operation on the parent-child relations till a fixed-point is reached. It has been shown that processing such queries is unlikely to be practical [AJKP02].

Since XML documents are ordered, labeled trees, there are several proposals on encoding the order of XML data so that the queries can exploit the order information. Zhang et al. [ZND+01] proposed the "inverted list" encoding method that stores XML nodes by giving each node a starting position, ending position and nesting depth (which will be explained in detail in the next section). With this encoding, queries like ancestor-descendant and parent-child relationship can be presented naturally using RDBMS. More interestingly, the ancestor-descendant relationship is not harder (even simpler) to be processed than the parent-child relationship, which is contrary to the folklore theory that ancestor-descendant queries always involve recursions and are more complicated to deal with.

As the basic component for most XML query languages, XPath [CD99,MS02] is the focus of many theoretical research activities. Gottlob and Koch [GK02b] showed that query complexity of XPath is in linear time. The underlying theory is monadic datalog on trees. Miklau and Suciu [MS02] proved that the query containment of a fragment of XPath is Co-NP complete and the containment checking algorithm is based on the transformation from XPath to tree automata.
8. QUERY EQUIVALENCE OVER TREES

Considering the "inverted list" encoding method on XML data and the XPath queries on it, several questions arise naturally: can XPath queries be expressed with relational queries (a.k.a. conjunctive queries with some extensions) in this framework? Under this scenario, are there any efficient algorithm for containment checking? What is the difference of the containment checking on XML trees, in comparison with the normal containment checking? In this chapter, we give the answers to the above questions.

8.1.1 Contributions

We show that XPath queries form a fragment of Conjunctive Queries with Linear Constraints on Trees (CQLC' ) and address the containment checking problem on such queries. We make the following contributions:

- In the context of containment checking of queries we show that the homomorphism property still holds on queries with child, descendant and following axes. As a result, a polynomial time complexity is obtained for this fragment.

- The containment problem that we consider in this work is however restricted to tree databases. In order to find a method for tree containment checking, we give a wildcards elimination algorithm, which first calculates the closure of the constraints by a set of axioms and then eliminates the related information of wildcard nodes. We prove that the query after the wildcards elimination processing is nevertheless tree equivalent to the original query and thus the general containment checking algorithm can be applied.

- We show that several existing results on conjunctive queries with linear constraints can be exploited to analyze the XML queries. Due to the tree model properties of XML databases, specialized algorithms have to be considered. Nevertheless the results we obtained are of interest in their own right.

The rest of the chapter is organized as follows: In Section 8.2 we introduce the "inverted list" encoding on XML data and give the translation of XPath queries to conjunctive queries with linear constraints. Section 8.3 mainly deals with the containment problem. Finally in Section 8.5 the conclusions are presented.

8.1.2 Related Work

Ordered XML encoding has been intensively studied recently. Tatarinov et al. [TVB+02] proposed three encoding schema, namely global order, local (sibling) order and Dewey order and compared them on a workload of ordered XML queries and updates. Some other encoding methods are discussed in [SSK+01, FK99].
8.2. PRELIMINARIES

Regarding the containment checking of XML queries, there are several semantics being applied. These are: the tree pattern method and regular expressions method [MS02, ACLS01], the monadic datalog on trees method [GK02b], and the tree automata method [NS03, MS02].

Complexity of containment checking for several extensions of XPath are proposed as well [NS03, Woo03, CGV03].

To the best of our knowledge, there is no previous work on containment checking for XML queries using conjunctive queries with linear constraints on trees.

Containment checking of conjunctive queries with arithmetic comparison and linear constraints has been extensively investigated in the database community [Klu88, ZO94, Ull97]. Constraint databases as a generalized form of conjunctive queries with constraints, have been intensively studied as well [KLP00].

8.2 Preliminaries

8.2.1 Modeling XML data with inverted lists

An XML database is a forest of rooted, ordered and labeled trees. Each node corresponds to an element and the edge represents the element/subelement relationship. In order to capture the document order of XML data, there have been several encoding mechanisms to "shred" XML data in the relational database. Zhang et al. [ZND+01] proposed the "inverted list" method by assigning for each node a start position, an end position, and a nesting depth.

The inverted list encoding method works as follows: there are two indexes: \( E\text{-index} \), storing indexed elements and \( T\text{-index} \), storing indexed text words. Each inverted list records the occurrences of an element or a word. Each occurrence is indexed by it document number, its position and its nesting depth within the document. The example in Figure 8.1 illustrates this. Observe that for each element in the XML excerpt (Figure 8.1(a)), there is a 3-tuple \( (docno, begin : end, level) \) associated with it, where \( docno \) is the identifier for the document, \( begin \) and \( end \) denote the position of the element node generated by counting the word numbers in the XML tree with a depth-first traversal. Since each non-leaf node is always traversed twice, the \( begin \) position records the number before all its children are visited and the \( end \) position records the number after that. The \( level \) tag denotes the nesting depth of the node within the document. Similarly for each word node, a 3-tuple \( (docno, wordno, level) \) is used to represent it. The only difference between the element node and word node is that text words are all leaf nodes, so that they are
8. QUERY EQUIVALENCE OVER TREES

<?xml version="1.0"?>
<product>
  <category>vegetables</category>
  <item>Japanese Eggplant</item>
  <inventory>
    <sku>Japavegequh409</sku>
    <price>0.13</price>
    <inventory>720</inventory>
  </inventory>
  <vendor>TriCountry Produce</vendor>
</product>

(a)

<vendor>TriCountry Produce</vendor>

(b)

Figure 8.1: A sample XML document fragment (a) and the corresponding tree representation (b).

traversed only once.

8.2.2 Translation of XPath Axes

E-index and T-index can be mapped into the following two relations [ZND+01]:

\[
\text{ELEMENTS}(\text{term, docno, begin, end, level})
\]

\[
\text{TEXTS}(\text{term, docno, wordno, level})
\]

Note that firstly, since the join algorithm of \textit{element/text} is nearly identical as that one of \textit{element/element}, we will only consider the ELEMENTS relation in the remainder of the chapter. Secondly, since the equi-join of \textit{docno} is trivial, and does not affect the query evaluation, we will leave it out from the relation schema either. As a result, the database schema contains one table:

\[
e(\text{term, begin, end, level})
\]

Given the XML data encoded with the "inverted list", the translation of XPath axes to conjunctive queries with linear constraints is straightforward:

- **descendant::**  (//)
  a tree node \( n_2 \) whose position in the XML database is encoded as \( e(n_2, B_2, E_2, L_2) \) is a descendant of a tree node \( n_1 \) whose position is encoded as \( e(n_1, B_1, E_1, L_1) \) iff \( B_1 < B_2, E_2 < E_1 \) and \( L_1 < L_2 \).

- **child::**  (/)

8.3. CONTAINMENT CHECKING OF $\mathcal{CQLC}^T$

a tree node $n_2$ whose position in the XML database is encoded as $e(n_2, B_2, E_2, L_2)$ is a child of a tree node $n_1$ whose position is encoded as $(n_1, B_1, E_1, L_1)$ iff $B_1 < B_2$, $E_2 < E_1$, and $L_1 + 1 = L_2$.

- **parent:**
a tree node $n_2$ whose position in the XML database is encoded as $e(n_2, B_2, E_2, L_2)$ is the parent of a tree node $n_1$ whose position is encoded as $e(n_1, B_1, E_1, L_1)$ iff $B_2 < B_1$, $E_1 < E_2$, and $L_2 + 1 = L_1$.

- **ancestor:**
a tree node $n_2$ whose position in the XML database is encoded as $e(n_2, B_2, E_2, L_2)$ is the ancestor of a tree node $n_1$ whose position is encoded as $e(n_1, B_1, E_1, L_1)$ iff $B_2 < B_1$, $E_1 < E_2$ and $L_2 < L_1$.

- **following:**
a tree node $n_2$ whose position in the XML database is encoded as $e(n_2, B_2, E_2, L_2)$ is following of a tree node $n_1$ whose position is encoded as $e(n_1, B_1, E_1, L_1)$ iff $B_2 > E_1$.

- **preceding:**
a tree node $n_2$ whose position in the XML database is encoded as $e(n_2, B_2, E_2, L_2)$ is preceding of a tree node $n_1$ whose position is encoded as $e(n_1, B_1, E_1, L_1)$ iff $E_2 < B_1$.

- **following-sibling:**
a tree node $n_2$ whose position in the XML database is encoded as $e(n_2, B_2, E_2, L_2)$ is following-sibling of a tree node $n_1$ whose position is encoded as $e(n_1, B_1, E_1, L_1)$ iff $B_2 > E_1$ and it shares the same parent with $n_1$.

- **preceding-sibling:**
a tree node $n_2$ whose position in the XML database is encoded as $e(n_2, B_2, E_2, L_2)$ is preceding of a tree node $n_1$ whose position is encoded as $e(n_1, B_1, E_1, L_1)$ iff $E_2 < B_1$ and it shares the same parent with $n_1$.

8.3 Containment Checking of $\mathcal{CQLC}^t$

In the following section, we give a formal definition of XPath queries containing the axes listed above, on ordered XML data which is encoded using "inverted list". The queries turn out to be a fragment of conjunctive queries with linear constraints ($\mathcal{CQLC}^t$). After a careful observation, we show that the queries have the property of query width, which immediately results in a polynomial time query processing complexity. Similar as on finite words, a fragment of the queries, namely the queries
with axes child, descendant and following has the homomorphism property, and consequently the containment checking is in polynomial time.

8.3.1 Query Evaluation of $CQLC^t$

The translated XPath queries with standard axes, which are called $CQLC^t$ queries, have the following form:

$$h() : e(T_1, B_1, E_1, L_1), e(T_2, B_2, E_2, L_2), \ldots, e(T_n, B_n, E_n, L_n),$$

$$\beta^{BE}, \beta^L.$$  

Where each node in the query is represented by a predicate $e$ with the variables $T, B, E, L$ standing for term, begin, end, and level, respectively. Terms can either be a constant string or a distinct variable, representing the wildcard. $\beta^{BE}$ indicates the conjunction of linear constraints concerning begin and end variables, and $\beta^L$ the conjunction of linear constraints concerning the level variables.

Note that besides the constraints listed explicitly in the translation rules, the constraint IC1 as follows holds on all the $CQLC^t$ queries.

$$B_i < E_i$$

$$B_i < B_j \land B_j < E_j \land E_j < B_i \rightarrow L_i < L_j$$

The above integrity constraints should be added into any translated queries. However, to simplify the representation of the queries, we do not write those constraints which can be implied from the others.

**Example 26** Consider the following XPath query $Q^{XP}$:

$$Q^{XP} : \text{a/desc ::a/following ::a/desc ::a/child ::a}$$

The translated $CQLC^t$ query is as follows:

$$Q : q \rightarrow e(a, B_1, E_1, L_1),$$

$$e(a, B_2, E_2, L_2), B_1 < B_2, E_2 < E_1, B_2 < E_2, L_1 < L_2,$$

$$e(a, B_3, E_3, L_3), E_2 < B_3,$$

$$e(a, B_4, E_4, L_4), B_3 < B_4, E_4 < E_3, L_3 < L_4,$$

$$e(a, B_5, E_5, L_5), B_4 < B_5, E_5 < E_4, B_5 < E_5, L_4 + 1 = L_5. \Box$$

 Conjunctive queries with linear constraints (CQLC) are widely used in every database application. One needs often arithmetic comparisons such as (price < 50) or (x + 5 = y) expressing the SPJ (select-projection-join) queries. The following is the formal definition of conjunctive queries with linear constraints.
8.3. CONTAINMENT CHECKING OF CQLC⁻

Definition 23 A conjunctive query with linear constraints (CQLC) has the following form:

\[ Q_2: h \models G_1, \ldots, G_k, F_1, \ldots, F_m. \]

where each \( G_i (1 \leq i \leq k) \) has the form \( r_i(X_i) \), where \( r_i \) is a predicate (relation name) in a given database schema, and \( X_i \) is a list of terms (variables or constants). Each \( G_i (1 \leq i \leq k) \) is an ordinary atom or an ordinary subgoal. \( F_i (1 \leq i \leq m) \) is an atomic constraint with one of the following forms:

\[ X \theta Y, \quad X \theta c, \quad X + c \theta Y \]

where \( \theta \in \{ \neq, <, >, \leq, \geq, = \} \). We assume that all the variables in \( F_1, \ldots, F_m \) have values over (discrete) totally ordered domains, such as integer. Furthermore, it is required that every variable occurs in the constraints appears in the ordinary atoms: \( \text{Var}(F_1, \ldots, F_m) \subseteq \text{Var}(G_1, \ldots, G_k) \).

Clearly the query evaluation of general CQLC queries is at least as hard as CQs, which is a special fragment of CQLC queries with only ordinary atoms. Since acyclic CQs have polynomial time query evaluation complexity (combined complexity), it is interesting to know whether the CQLC⁻ queries, a fragment of general CQLC queries, are acyclic. The hypergraph of any CQLC query can be drawn in the similar way as those of CQs. One has to only consider each atomic constraint as an ordinary atom. It is obvious that the hypergraph of any CQLC⁻ query is cyclic.

However, one may notice that the cyclicity is local. In fact, the variables \( T_i, B_i, E_i, L_i \) of the subgoal \( e(T_i, B_i, E_i, L_i) \) are only related to the variables in the subgoal before it: \( e(T_{i-1}, B_{i-1}, E_{i-1}, L_{i-1}) \) and the subgoal after it: \( e(T_{i+1}, B_{i+1}, E_{i+1}, L_{i+1}) \). Since the number of atomic constraints on the begin, end and level variables between two adjacent subgoals (e.g. \( e(T_i, B_i, E_i, L_i) \) and \( e(T_{i+1}, B_{i+1}, E_{i+1}, L_{i+1}) \)) are bounded, the query decomposition of each CQLC⁻ query has a bounded width. Figure 8.2 illustrates a 4-width query decomposition of query in Example 26.

Lemma 8 Every CQLC⁻ query \( Q \) has a bounded query width.

The polynomial time upper bound of query containment for acyclic CQs was first proposed by Yannakakis [Yan81]. Chekuri and Rajaraman [CR97] extended the results of Yannakakis by proposing an algorithm for the containment checking

\( ^1 \)The only exceptions are \( : \text{following-sibling} \) and \( : \text{preceding-sibling} \), where a common parent node has to be generated. However, it is obvious that the locality on subgoals remains.
of acyclic CQs and the corresponding soundness and completeness proof. Furthermore, Chekuri and Rajaraman pointed out that for CQs with bounded query width, the containment checking is in polynomial time as well, and the acyclic CQ is a conjunctive query with bounded query width 1. The algorithm of containment checking of CQs with bounded query width was also proposed, with the basic principle being same as that of acyclic CQs. Since the containment checking for both CQs with bounded query width and acyclic CQs are in polynomial time, we abuse the term "acyclic CQ" also to represent "CQ with bounded query width".

**Theorem 18 (Query evaluation of acyclic CQ)** [Yan81, CR97] The query evaluation of acyclic conjunctive queries (combined complexity) is in polynomial time. \(\square\)

**Lemma 9 (Query evaluation of CQLC')** The query evaluation (combined complexity) of CQLC' queries on ordered XML is in polynomial time. \(\square\)

### 8.3.2 Containment

Generally there are two methods checking containment of CQLC queries \(Q_1 \sqsubseteq Q_2\):

1. Using canonical databases [Klu88, Ull97, LaSS93]. If the domain of the involved variables has an order, one can generate an exponential number of canonical
8.3. CONTAINMENT CHECKING OF CQLC$^t$

databases (a.k.a. representative databases in [Klu88]) of $Q_1$, and for each such canonical database $D$, one applies $Q_2$ on it. If $Q_2(D)$ returns true for every canonical database, the containment $Q_1 \subseteq Q_2$ holds. Otherwise, $Q_1 \not\subseteq Q_2$. We shall use this method to give an upper bound for the general containment checking of CQLC$^t$ queries, without much technical details.

2. Containment mapping method. This method has more practical value and in many cases more efficient than the former one [ALM04,ZÖ97]. In the later part of the section, we use this method to prove the homomorphism property of a fragment of CQLC$^t$ queries.

Given the polynomial time query evaluation of CQLC$^t$ queries in Lemma 9, a Co-NP upper bound of containment checking is obvious. Because there is an exponential number of canonical databases to be tested, and each time the evaluation is in polynomial time.

**Lemma 10 (Containment of CQLC$^t$ queries)** The containment of CQLC$^t$ queries is in Co-NP$^2$.

---

**Homomorphism**

For conjunctive queries$^3$, the so-called homomorphism property (a.k.a. containment mapping) always holds: given two CQs $Q_1$ and $Q_2$, $Q_1 \subseteq Q_2$ if and only if there is a homomorphism (or containment mapping) from subgoals in $Q_2$ to those in $Q_1$. However, the homomorphism does not hold anymore if the linear constraints are allowed in conjunctive queries. Although a containment mapping on the ordinary subgoals, coupled with the condition that the constraints of $Q_1$ imply the mapped constraints of $Q_2$ is sufficient to show $Q_1 \subseteq Q_2$. However, it is not necessary.

**Theorem 19** [Ull89] Let $Q_1$ and $Q_2$ be two CQLC queries:

$$Q_1 : h \leftarrow j_1, \ldots, j_i, k_1, \ldots, k_n.$$  

$$Q_2 : h \leftarrow g_1, \ldots, g_k, f_1, \ldots, f_m.$$  

Where the $j$’s and $g$’s are ordinary subgoals and the $k$’s and $f$’s are atomic constraints. Let the following condition hold:

1. There is a containment mapping $\rho$ from the variables of $Q_2$ to the variables of $Q_1$ such that every ordinary subgoal of $Q_2$ is turned into an ordinary subgoal of $Q_1$.

---

$^2$Whether the bound is tight is unknown. We conjecture that it is Co-NP hard.

$^3$We use this term, or simply CQs, to denote conjunctive queries without any extension.
2. For each \( i = 1, 2, \ldots, m, \rho(F_i) \) is logically implied by the constraints of \( Q_1 \), which can be formally written as follows:
\[
(K_1 \land \ldots \land K_n) \rightarrow (\rho(F_1) \land \ldots \land \rho(F_m))
\]

Then \( Q_1 \models Q_2 \).

### Terminology on logical implication of linear constraints

In the following we introduce some technical observation on the logical implication. Assume the domain of all the variables in the atomic constraints is integer. Note that the following results can be naturally extended to rational and real domains.

An assignment \( \pi \) to a variable \( X \), denoted as \( \pi(X) \), maps \( X \) to a member of its domain (possibly infinite). An assignment \( \pi \) satisfies an atomic constraint \( \alpha \), if \( \pi(\alpha) = \text{true} \) according to the properties of integers.

A conjunction of atomic constraints \( \alpha_1 \land \ldots \land \alpha_k (k \geq 1) \) logically implies an atomic constraint \( \beta \), denoted as \( \alpha_1 \land \ldots \land \alpha_k \rightarrow \beta \), if \( \text{Var}(\beta) \subseteq \text{Var}(\alpha_1, \ldots, \alpha_k) \), and every assignment \( \pi \) which satisfies \( \alpha \), also satisfies \( \beta \). On the other hand, \( \alpha_1 \land \ldots \land \alpha_k \rightarrow \beta \), if there exists one assignment \( \pi \) which satisfies \( \alpha \), but does not satisfy \( \beta \).

The negated form of an atomic constraint \( \alpha \) is denoted as \( \neg \alpha \). \( \neg \alpha \) can be obtained by replacing the boolean operation \( \theta \) in \( \alpha \) with its complementary operation \( \theta' \), while the other parts remain unchanged. For instance, let \( \alpha = X < Y \), then \( \neg \alpha = X \geq Y \). Given any atomic constraint \( \alpha \), it is obvious that if any assignment \( \pi \) satisfies \( \alpha \), then \( \pi \) does not satisfy \( \neg \alpha \).

Unfortunately, the conditions of Theorem 19 are not necessary for a containment to hold between two conjunctive queries with linear constraints. There can be interactions between the ordinary and constraints that make condition (2) unnecessary.

**Example 27** Consider the CQLC queries \( Q_1 \) and \( Q_2 \) as follows:

\[
\begin{align*}
Q_1 & : \ h \ :- \ r(U, V), r(V, U). \\
Q_2 & : \ h \ :- \ r(A, B), A \leq B.
\end{align*}
\]

Obviously condition (2) of Theorem 19 does not hold, because there are no constraints of \( Q_1 \) to imply \( \rho(U \leq V) \). However, the containment relation \( Q_1 \models Q_2 \) holds.

Zhang and and Özsoyoglu [ZÖ97] proposed a necessary and sufficient condition for the containment checking of conjunctive queries with constraints.
8.3. CONTAINMENT CHECKING OF CQLC$^T$

**Theorem 20** Let $Q_1$ and $Q_2$ be two CQLCs:

$Q_1: \ h \ := \ J_1, \ldots, J_i, K_1, \ldots, K_n.$

$Q_2: \ h \ := \ G_1, \ldots, G_k, F_1, \ldots, F_m.$

Where the $J$’s and $G$’s are ordinary subgoals and the $K$’s and $F$’s are atomic constraints. Let $\rho_1, \rho_2, \ldots, \rho_r$ be all the containment mappings from the variables of $Q_2$ to the variables of $Q_1$, such that every ordinary subgoal of $Q_2$ is turned into an ordinary subgoal of $Q_1$.

Then $Q_1 \subseteq Q_2$ if and only if the following holds:

\[(K_1 \land \ldots \land K_n) \rightarrow \bigvee_{i=1}^{r} (\rho_i(F_1) \land \ldots \land \rho_i(F_m)) \quad (8.1)\]

Now consider Example 27, there are two containment mappings from the variables of ordinary subgoals in $Q_2$ to those of the ordinary subgoals in $Q_1$, namely $\rho_1: \{A \rightarrow U, B \rightarrow V\}$ and $\rho_2: \{A \rightarrow V, B \rightarrow U\}$. Since $true \rightarrow (\rho_1(U \leq V) \lor \rho_2(U \leq V))$, the containment $Q_1 \subseteq Q_2$ holds.

Given two queries $Q_1$ and $Q_2$, and all the containment mappings from the variables of ordinary subgoals in $Q_2$ to those of the ordinary subgoals in $Q_1$, there are some containment mappings which do not influence the implication test. We call such containment mappings "invalid mappings".

**Definition 24 (Invalid Mapping)** Let $Q_1$ and $Q_2$ be two conjunctive queries with built-in subgoals:

$Q_1: \ h \ := \ J_1, \ldots, J_i, K_1, \ldots, K_n.$

$Q_2: \ h \ := \ G_1, \ldots, G_k, F_1, \ldots, F_m.$

Where the $J$’s and $G$’s are ordinary subgoals and the $K$’s and $F$’s are atomic constraints. Let $\rho_1, \rho_2, \ldots, \rho_r$ be all the containment mappings from the variables of $Q_2$ to the variables of $Q_1$, such that every ordinary subgoal of $Q_2$ is turned into an ordinary subgoal of $Q_1$.

The containment mapping $\rho_u(1 \leq u \leq r)$ is invalid, if the following holds:

1. \[(K_1 \land \ldots \land K_n) \rightarrow (\rho_u(F_1) \land \ldots \land \rho_u(F_m)) \quad \square\]

2. \[(K_1 \land \ldots \land K_n) \rightarrow \bigvee_{i=1}^{r} (\rho_i(F_1) \land \ldots \land \rho_i(F_m))\]
8. QUERY EQUIVALENCE OVER TREES

iff

\[
(K_1 \land \ldots \land K_n) \rightarrow \bigvee_{i=1}^{u-1} (\rho_i(F_1) \land \ldots \land \rho_i(F_m)) \lor \bigvee_{i=u+1}^{r} (\rho_i(F_1) \land \ldots \land \rho_i(F_m))
\]

Intuitively, a containment mapping is invalid if it does not contribute to the implication test. Before we give the syntax restriction on the containment mappings which are invalid, we shall introduce some technical observation.

The right hand side (rhs) of 8.1 is a disjunction and each disjunct is a conjunction of atomic constraints. By applying the distributive law, it can be equivalently written into a conjunction and each conjunct is a disjunction of atomic constraints. For instance, \((X < Y \land A < B) \lor (X < Y \land A \geq B)\) can be equivalently written as

\[
(X < Y \lor X < Y) \land (X < Y \lor A \geq B) \land (A < B \lor X < Y) \land (A < B \lor A \geq B)
\]

As a result, 8.1 can be equivalently written as

\[
(K_1 \land \ldots \land K_n) \rightarrow \bigwedge_{i=1}^{m'} (\rho_1(F_{i,1}) \lor \ldots \lor \rho_r(F_{i,r})) (F_{i,j} : 1 \leq j \leq r) \in \{F_1, \ldots, F_m\}
\]

(8.2)

Clearly 8.1 and 8.2 are equivalent, the size of the rhs of 8.2 is exponential to that of 8.1. To simplify the representation, we name 8.1 as DNF (Disjunctive Normal Form) implication, and 8.2 as CNF (Conjunctive Normal Form) implication. Now we are ready to identify those containment mappings which are invalid.

**Theorem 21** Let \(Q_1\) and \(Q_2\) be two conjunctive queries with linear constraints:

\[
Q_1 : \ h \ := \ J_1, \ldots, J_t, K_1, \ldots, K_n.
Q_2 : \ h \ := \ G_1, \ldots, G_k, F_1, \ldots, F_m.
\]

Where the \(J\)'s and \(G\)'s are ordinary subgoals and the \(K\)'s and \(F\)'s are atomic constraints. Let \(\rho_1, \rho_2, \ldots, \rho_r\) be all the containment mappings from the variables of \(Q_2\) to the variables of \(Q_1\) such that every ordinary subgoal of \(Q_2\) is turned into an ordinary subgoal of \(Q_1\).

The containment mapping \(\rho \in \{\rho_1, \ldots, \rho_r\}\) is invalid, if

1. There is an \(i(1 \leq i \leq m)\), such that \(\rho(F_i) \rightarrow false\), or
8.3. CONTENTMENT CHECKING OF CQLC$^T$

2. There is a set of atomic constraints \( \{F'_1, \ldots, F'_j\} \subseteq \{F_1, \ldots, F_m\} \), where \( j \leq m \), such that \( \rho(F'_1) \land \ldots \land \rho(F'_j) \rightarrow \text{false} \), or

3. There is an \( i (1 \leq i \leq m) \), such that \( \rho(F_i) \rightarrow \neg \phi \), where \( \phi \) is an atomic constraint and \( K_1 \land \ldots \land K_n \rightarrow \phi \), or

4. There is a set of atomic constraints \( \{F'_1, \ldots, F'_j\} \subseteq \{F_1, \ldots, F_m\} \), where \( j \leq m \), such that \( \rho(F'_1) \land \ldots \land \rho(F'_j) \rightarrow \neg \phi \). Where \( \phi \) is an atomic constraint and \( K_1 \land \ldots \land K_n \rightarrow \phi \).

\[ \square \]

**Example 28** Consider the following queries:

\[
Q_1 : \quad q := r(A, D), r(B, E), r(C, F), A < B, B < C.
\]

\[
Q_2 : \quad q := r(X, U), r(Y, V), X < Y.
\]

There are \( 3^2 \) containment mappings from the variables in \( Q_2 \) to those in \( Q_1 \). Because there are 3 and 2 ordinary subgoals in \( Q_1 \) and \( Q_2 \) respectively. The mappings are:

\[
\rho_1 : \quad \{X \rightarrow A, U \rightarrow D, Y \rightarrow A, V \rightarrow D\}
\]

\[
\rho_2 : \quad \{X \rightarrow A, U \rightarrow D, Y \rightarrow B, V \rightarrow E\}
\]

\[
\rho_3 : \quad \{X \rightarrow A, U \rightarrow D, Y \rightarrow C, V \rightarrow F\}
\]

\[
\rho_4 : \quad \{X \rightarrow B, U \rightarrow E, Y \rightarrow A, V \rightarrow D\}
\]

\[
\rho_5 : \quad \{X \rightarrow B, U \rightarrow E, Y \rightarrow B, V \rightarrow E\}
\]

\[
\rho_6 : \quad \{X \rightarrow B, U \rightarrow E, Y \rightarrow C, V \rightarrow F\}
\]

\[
\rho_7 : \quad \{X \rightarrow C, U \rightarrow F, Y \rightarrow A, V \rightarrow D\}
\]

\[
\rho_8 : \quad \{X \rightarrow C, U \rightarrow F, Y \rightarrow B, V \rightarrow E\}
\]

\[
\rho_9 : \quad \{X \rightarrow C, U \rightarrow F, Y \rightarrow C, V \rightarrow F\}
\]

Among the 9 containment mappings, only \( \rho_2 \) and \( \rho_3 \) are valid mappings, and the rest of the mappings are invalid, which means that they do not influence the result of implication test. To see this, consider \( \rho_1 \), we obtain that \( \rho_1(X < Y) = A < A \), which implies \text{false}. This is defined in condition (1) in Theorem 21. As a result, \( \rho_1 \) is an invalid mapping. Similarly, \( \rho_2(X < Y) = B < A \). Let \( \phi = A < B \), then \( \neg \phi = A \geq B \). It is obvious that \( \rho_4(X < Y) \rightarrow \neg \phi \). This satisfies the condition (3) in Theorem 21. Consequently, \( \rho_4 \) is an invalid mapping too.

\[ \square \]

The proof of Theorem 21 is as follows.

**Proof 10** We have to prove that the containment mapping \( \rho \in \{\rho_1, \ldots, \rho_r\} \) is invalid (cf. Definition 24), if one of the conditions (1) - (4) holds.

Since the sequence of applying the disjunction of the mappings is irrelevant, we assume that \( \rho = \rho_r \).
8. QUERY EQUIVALENCE OVER TREES

One has to show that

\[(K_1 \land \ldots \land K_n) \rightarrow \bigvee_{i=1}^{r} (\rho_i(F_1) \land \ldots \land \rho_i(F_m))\]

iff

\[(K_1 \land \ldots \land K_n) \rightarrow \bigvee_{i=1}^{r-1} (\rho_i(F_1) \land \ldots \land \rho_i(F_m))\]

"If": trivial.
"Only if": Assume that

\[(K_1 \land \ldots \land K_n) \rightarrow \bigvee_{i=1}^{r-1} (\rho_i(F_1) \land \ldots \land \rho_i(F_m))\]

(8.3)

we have to prove that

\[(K_1 \land \ldots \land K_n) \rightarrow \bigvee_{i=1}^{r} (\rho_i(F_1) \land \ldots \land \rho_i(F_m))\]

Consider the CNF implication of 8.3. We obtain that there exist \(l (l \geq 1)\) formulas \(\alpha_1, \ldots, \alpha_l\), s.t.

\[(K_1 \land \ldots \land K_n) \rightarrow \bigwedge_{i=1}^{l} (\alpha_i)\]

(8.4)

where each \(\alpha_i (1 \leq i \leq l)\) is the disjunction of \(r-1\) atomic constraints.

Now we make the disjunction operation between \(\rho_r(F_1) \land \ldots \land \rho_r(F_m)\) and the rhs of 8.4:

\[\bigwedge_{i=1}^{l} (\alpha_i) \lor \bigwedge_{j=1}^{m} (\rho_r(F_j))\]

(8.5)

which is equal to:

\[\bigwedge_{i=1}^{l} \bigwedge_{j=1}^{m} (\alpha_i \lor \rho_r(F_j))\]

(8.6)

We have to prove that

\[(K_1 \land \ldots \land K_n) \rightarrow \bigwedge_{i=1}^{l} \bigwedge_{j=1}^{m} (\alpha_i \lor \rho_r(F_j))\]

under each of the 4 conditions as follows:
1. Let $\rho_r(F_k) \rightarrow false$ where $(1 \leq k \leq m)$. Since

$$ (K_1 \land \ldots \land K_n) \rightarrow \alpha_i $$

holds, there exists one assignment $\pi$ satisfying $(K_1 \land \ldots \land K_n)$ but does not satisfy $\alpha_i$. Because there is no assignment satisfying $\rho_r(F_k)$, We obtain that for each $i(1 \leq i \leq l)$,

$$ (K_1 \land \ldots \land K_n) \rightarrow (\alpha_i \lor \rho_r(F_k)) $$

holds.

2. W.l.o.g. we assume there is a set of atomic constraints $\{F_1, \ldots, F_j\} \subseteq \{F_1, \ldots, F_m\}$, where $j \leq m$, such that $\rho_r(F_1) \land \ldots \land \rho_r(F_j) \rightarrow false$.

We obtain that for each $i(1 \leq i \leq l)$, there is at least one $u(1 \leq j \leq u)$, such that

$$ (K_1 \land \ldots \land K_n) \rightarrow (\alpha_i \lor \rho_r(F_u)) $$

Because otherwise, the following would hold:

$$ (K_1 \land \ldots \land K_n) \rightarrow (\alpha_i \lor (\rho_r(F_1) \land \ldots \land \rho_r(F_j))) $$

which means, there is one assignment $\pi$ satisfies $K_1 \land \ldots \land K_n$, but does not satisfy $\alpha_i$, however satisfies $\rho_r(F_1) \land \ldots \land \rho_r(F_j)$. This is a contradiction to $\rho_r(F_1) \land \ldots \land \rho_r(F_j) \rightarrow false$.

3. Let $\rho_r(F_k) \rightarrow \neg \phi$ where $(1 \leq k \leq m)$, then we obtain that for each $i(1 \leq i \leq l)$,

$$ (K_1 \land \ldots \land K_n) \rightarrow (\alpha_i \lor \rho_r(F_k)) $$

Let $\pi$ be any assignment satisfying $K_1 \land \ldots \land K_n$, but does not satisfy $\alpha_i$. Because $K_1 \land \ldots \land K_n \rightarrow \phi$, then $\pi$ satisfies $\phi$. It follows that $\pi$ does not satisfy $\neg \phi$. Since $\rho_r(F_k) \rightarrow \neg \phi$, $\pi$ does not satisfy $\rho_r(F_k)$, because otherwise, $\pi$ would have to satisfy $\neg \phi$, which is a contradiction.

4. can be proven by combining condition (2) and (3).

\[\Box\]

**Definition 25 (Homomorphism)** Let $Q_1$ and $Q_2$ be two CQLCs:

- $Q_1: h \rightarrow J_1, \ldots, J_l, K_1, \ldots, K_n.$
- $Q_2: h \rightarrow G_1, \ldots, G_k, F_1, \ldots, F_m.$

Where the $J$'s and $G$'s are ordinary subgoals and the $K$'s and $F$'s are atomic constraints.

The homomorphism property holds if the following holds: $Q_1 \subseteq Q_2$ if and only if there is a containment mapping $\rho$ from variables of $Q_2$ to the variables of $Q_1$ such that every ordinary subgoal of $Q_2$ is turned into an ordinary subgoal of $Q_1$, and

$$ (K_1 \land \ldots \land K_n) \rightarrow (\rho(F_1) \land \ldots \land \rho(F_m)) $$
Given the acyclic property of the CQs with constraints we have discussed, it is interesting to consider the relationship between the homomorphism and the acyclic properties. Clearly, the homomorphism property does not imply the acyclicity. One can easily give two queries which are left semi-interval – consequently the homomorphism property holds [Klu88]– but the ordinary predicates does not have an acyclic hypergraph. On the other hand, we show in the following example that the acyclicity property of the queries does not imply the homomorphism property either.

Example 29 Given two queries \( Q_1 \) and \( Q_2 \) as follows:

\[
Q_1 : \quad q \iff r(U), r(V), U \neq V.
Q_2 : \quad q \iff r(A), r(B), A < B.
\]

It is obvious that both queries are acyclic. In fact, \( Q_1 \) and \( Q_2 \) are contained in each other. However, to prove that \( Q_1 \sqsubseteq Q_2 \), one has to find all containment mappings from \( Q_2 \) to \( Q_1 \), namely \( \rho_1 : A\rightarrow U, B\rightarrow V \) and \( \rho_2 : A\rightarrow V, B\rightarrow U \). As a result, the implication relation:

\[
U \neq V \rightarrow \rho_1(A < B) \lor \rho_2(A > B)
\]

holds, since it is equivalent to the following formula:

\[
U \neq V \rightarrow (U < V \lor U > V)
\]

Note that this implication does not hold with any single containment mapping from \( Q_2 \) to \( Q_1 \). \( \Box \)

Homomorphism Property of \( CQ\text{LC}^t \)

To simplify the presentation of the problem, we introduce several notations. Given \( Q \) a \( CQ\text{LC}^t \) query, \( \text{core}(Q) \) is used to denote conjunction of ordinary subgoals of \( Q \), \( \beta^\text{BE} \) indicates the linear constraints concerning begin and end variables, and \( \beta^L \) the linear constraints concerning the level variables. As mentioned before, general conjunctive queries with linear constraints do not always have the homomorphism property. Next we prove that the fragment of \( CQ\text{LC}^t \) queries with child, descendant and following axes still has this nice property.

Theorem 22 (Homomorphism of \( CQ\text{LC}^t \)’s with forward axes) Given two \( CQ\text{LC}^t \) queries \( Q_1 \) and \( Q_2 \) with axes child, descendant and following as follows:

\[
Q_1 : \quad h() \iff e(T_1, B_1, E_1, L_1), \ldots, e(T_n, B_n, E_n, L_n), \beta_1^\text{BE}, \beta_1^L.
Q_2 : \quad h() \iff e(T'_1, B'_1, E'_1, L'_1), \ldots, e(T'_m, B'_m, E'_m, L'_m), \beta_2^\text{BE}, \beta_2^L.
\]
8.3. CONTAINMENT CHECKING OF $CQLC^T$

Figure 8.3: An invalid mapping

\[ e(T_1, B_1, E_1, L_1), \ldots, e(T_i, B_i, E_i, L_i), \ldots, e(T_n, B_n, E_n, L_n) \]

\[ e(T'_1, B'_1, E'_1, L'_1), \ldots, e(T'_m, B'_m, E'_m, L'_m), \ldots, e(T'_{n+1}, B'_{n+1}, E'_{n+1}, L'_{n+1}) \]

Figure 8.4: An invalid mapping

\[ e(T_1, B_1, E_1, L_1), \ldots, e(T_i, B_i, E_i, L_i), \ldots, e(T_n, B_n, E_n, L_n) \]

\[ e(T'_1, B'_1, E'_1, L'_1), \ldots, e(T'_m, B'_m, E'_m, L'_m), \ldots, e(T'_{n+1}, B'_{n+1}, E'_{n+1}, L'_{n+1}) \]

Figure 8.5: An invalid mapping

\[ e(T_1, B_1, E_1, L_1), \ldots, e(T_i, B_i, E_i, L_i), \ldots, e(T_n, B_n, E_n, L_n) \]

\[ e(T'_1, B'_1, E'_1, L'_1), \ldots, e(T'_m, B'_m, E'_m, L'_m), \ldots, e(T'_{n+1}, B'_{n+1}, E'_{n+1}, L'_{n+1}) \]

Figure 8.6: An invalid mapping

$Q_1 \subseteq Q_2$ holds if there is a containment mapping $\rho$ from variables in $Q_2$ to those in $Q_1$, such that

\[ \{ \rho(e(T'_1, B'_1, E'_1, L'_1)), \ldots, \rho(e(T'_m, B'_m, E'_m, L'_m)) \} \subseteq \{ e(T_1, B_1, E_1, L_1), \ldots, e(T_n, B_n, E_n, L_n) \}, \]

and further both $\beta^B_1 \rightarrow \rho(\beta^BE_2)$ and $\beta^L_1 \rightarrow \rho(\beta^L_2)$ hold.  

\[ \square \]
8. QUERY EQUIVALENCE OVER TREES

Proof 11 According to the results from Zhang and Özsoyoglu [ZÖ97], we obtain that $Q_1 \subseteq Q_2$ if and only if for all containment mappings from variables in $Q_2$ to those in $Q_1$: $\rho_1, \rho_2, \ldots, \rho_u$, and the following implication holds:

$$\beta_1^{BE} \land \beta_1^L \rightarrow \rho_1(\beta_2^{BE} \land \beta_2^L) \lor \ldots \lor \rho_u(\beta_2^{BE} \land \beta_2^L)$$

Now we have to prove that

$$\beta_1^{BE} \land \beta_1^L \rightarrow \rho_1(\beta_2^{BE} \land \beta_2^L) \lor \ldots \lor \rho_u(\beta_2^{BE} \land \beta_2^L)$$

if and only if there is a containment mapping $\rho$ from $\text{core}(Q_2)$ to $\text{core}(Q_1)$, such that $\beta_1^{BE} \rightarrow \rho(\beta_2^{BE})$ and $\beta_1^L \rightarrow \rho(\beta_2^L)$.

- "if": trivial. Assume that there is a containment mapping $\rho$ from variables in $Q_2$ to those in $Q_1$, such that $\beta_1^{BE} \rightarrow \rho(\beta_2^{BE})$ and $\beta_1^L \rightarrow \rho(\beta_2^L)$. Since $\beta_1^{BE}$ and $\beta_1^L$ do not share any common variable (which is also true for $\beta_2^{BE}$ and $\beta_2^L$), the following holds:

$$\beta_1^{BE} \land \beta_1^L \rightarrow \rho(\beta_2^{BE} \land \beta_2^L)$$

Since $\rho \in \{\rho_1, \ldots, \rho_u\}$, the following also holds:

$$\beta_1^{BE} \land \beta_1^L \rightarrow \rho_1(\beta_2^{BE} \land \beta_2^L) \lor \ldots \lor \rho_u(\beta_2^{BE} \land \beta_2^L)$$

- "only if": It is observed that all the begin variables in both $Q_1$ and $Q_2$ are totally ordered. We obtain that $B_1 < B_2 < \ldots < B_u$ and $B_1' < B_2' < \ldots < B'_u$. As a result, all the non-monotonic containment mappings from variables in $Q_2$ to those in $Q_1$ are invalid mappings. Assume that $B_u', B'_u$ are variables in $Q_2$ and $u < v$, further there is a containment mapping $\rho$, s.t. $\rho(B_u') = B_i$ and $\rho(B'_u) = B_j$, then we obtain $i < j$. All the invalid containment mappings, which are illustrated in Fig. 8.3, 8.4, 8.5 and 8.6 need not to be considered.

Let $\rho$ be the containment mapping illustrated in Fig. 8.3. $B_u'$ is mapped to $B_j$ and $B'_{u+1}$ is mapped to $B_i$. Let $\phi = B_i < B_j$. Clearly $\beta_1^{BE} \rightarrow \phi$. Since $\rho(B_u') = B_j < B_i$, and $B_j < B_i < B_u$, we have $\rho(\beta_2^{BE}) \rightarrow \phi$. This is proven to be a condition resulting in an invalid mapping in Theorem 21 (condition (3)). Similarly, other non-monotonic containment mappings can be proven invalid too.

Now we have to only consider all the monotonic containment mappings. One has to note that each containment mapping from variables in $Q_2$ to those in $Q_1$ has to turn each ordinary subgoal of $Q_2$ into one ordinary subgoals of $Q_1$. In fact, for each mapping $\rho$, only if

$$\{\rho(e(T_1', B'_1, E'_1, L'_1)), \ldots, \rho(e(T_m', B'_m, E'_m, L'_m))\} \subseteq \{e(T_1, B_1, E_1, L_1), \ldots, e(T_n, B_n, E_n, L_n)\},$$

holds, $\rho$ is called a containment mapping. This distinguishes the containment mapping from any arbitrary mapping.
8.3. CONTAINMENT CHECKING OF $CQLC^T$

As a result, given any containment mapping $\rho$, and variables $T'_u, B'_u, E'_u, L'_u$. If $\rho$ maps the variables as follows:

$$T'_u \rightarrow T_i, B'_u \rightarrow B_j, E'_u \rightarrow E_l, L'_u \rightarrow L_k$$

then $i = j = l = k$ holds. To see this, let a mapping $\rho$ map $B'_u$ to $B_i$ and $L'_u$ to $L_j$, where $i \neq j$. The mapped subgoal $e(T'_u, B'_u, E'_u, L'_u)$ has the form $e(\_, B_i, \_, L_j)$, where $i \neq j$, which would never be any subgoal in $Q_1$.

As a result, on end and level variables, the monotonic property holds too. Note that $i$ is not required that end and level variables be totally ordered. To see this, let $L'_u$ and $L'_v$ be level variables in $Q_2$, and $u < v$. Given a containment mapping $\rho$, s.t. $\rho(L'_u) = L_i$ and $\rho(L'_v) = L_j$. Since $\rho$ is a containment mapping, we have $\rho(B'_u) = B_i$ and $\rho(B'_v) = B_j$. Then we get $i < j$. The monotonic property on end variables can be proven similarly.

Besides those non-monotonic mappings, there are further containment mappings which can be easily identified as invalid. Given any adjacent subgoals $e(T_i, B_i, E_i, L_i)$ and $e(T_{i+1}, B_{i+1}, E_{i+1}, L_{i+1})$, we denote $e(T_i, B_i, E_i, L_i)$ as node $n_i$ and $e(T_{i+1}, B_{i+1}, E_{i+1}, L_{i+1})$ as node $n_{i+1}$ respectively. There are obviously 3 possible axes connecting them: child, descendant and following. Figure 8.7 to 8.10 illustrate that the mappings of a child or descendant axis to a following axis, as well as the mappings of a following axis to a child or descendant one, are invalid. Note that we use horizontal double line to represent following axes, a double line with 45 degree for descendant axes and a single line with 45 degree for child axes.

Let $\rho_1, \ldots, \rho_u$ be all the valid (monotonic) containment mappings from variables in $Q_2$ to those in $Q_1$, and let the following implication be true:

$$\beta_1^{BE} \land \beta_1^L \rightarrow \bigvee_{i=1}^{u} (\rho_i(\beta_2^{BE} \land \beta_2^L))$$
8. QUERY EQUIVALENCE OVER TREES

Figure 8.8: Mapping from a child axis to a following axis is an invalid mapping. The constraints with underlined variables cause a contradiction.

\[ B_j < E_j < B_{j+1} < E_{j+1} \]

\[ B'_i < B'_{i+1} < E'_{i+1} < E'_i, L'_i + 1 = L'_{i+1} \]

Figure 8.9: Mapping from a following axis to a descendant axis is an invalid mapping. The constraints with underlined variables cause a contradiction.

\[ B_j < B_{j+1} < E_{j+1} < E_{j}, L_j < L_{j+1} \]

\[ B'_i < E'_i < B'_{i+1} < E'_{i+1} \]

Figure 8.10: Mapping from a following axis to a child axis is an invalid mapping. The constraints with underlined variables cause a contradiction.

\[ B_j < B_{j+1} < E_{j+1} < E_{j}, L_j = L_{j+1} \]

\[ B'_i < E'_i < B'_{i+1} < E'_{i+1} \]

We have to prove that there exists one \( \rho_i(1 \leq i \leq u) \), such that

\[ \beta_i^{BE} \land \beta_i^L \to \rho_i(\beta_2^{BE} \land \beta_2^L) \]

holds.

Assume that there does not exist such a \( \rho_i(1 \leq i \leq u) \). This means that for each \( i(1 \leq i \leq u) \),

\[ \beta_i^{BE} \land \beta_i^L \to \rho_i(\beta_2^{BE} \land \beta_2^L) \]

This follows that for each \( i \), there is at least one atomic constraint \( \alpha_i \in \rho_i(\beta_2^{BE} \land \)
8.3. CONTAINMENT CHECKING OF $\mathcal{CQLC}^T$

$\beta_2^L$, s.t. $\beta_1^{BE} \land \beta_1^L \rightarrow \alpha_i$. We have to prove that

$$\beta_1^{BE} \land \beta_1^L \rightarrow \bigvee_{i=1}^{u} (\rho_i(\beta_2^{BE} \land \beta_2^L)) \quad (8.7)$$

Now consider the CNF form of the right-hand side of 8.7. We choose the conjunct with the form: $\alpha_1 \lor \ldots \lor \alpha_u$. Because according to the distributive law, this conjunct must exist. That is enough if we could prove that

$$\beta_1^{BE} \land \beta_1^L \rightarrow (\alpha_1 \lor \ldots \lor \alpha_u)$$

Note that we have for each $i$,

$$\beta_1^{BE} \land \beta_1^L \rightarrow \alpha_i$$

holds.

Now consider what kind of atomic constraints could exist in each $\alpha_i$. Since all the containment mappings are monotonic, each $\alpha_i(1 \leq i \leq u)$ could contain the following possible atomic constraints:

- $L_u < L_v$, \hspace{1cm} (u < v) \hspace{1cm} (1)
- $L_u + 1 = L_v$, \hspace{1cm} (u < v) \hspace{1cm} (2)
- $E_u > E_v$, \hspace{1cm} (u < v) \hspace{1cm} (3)
- $E_u < B_v$, \hspace{1cm} (u < v) \hspace{1cm} (4)

It might be noticed that such constraints as

- $B_u < B_v$, \hspace{1cm} (u < v) \hspace{1cm} (1')
- $B_u < E_u$ \hspace{1cm} (2')

are not included. This is because firstly, the \textit{begin} variables are totally ordered, so the constraints with the form as (1') are always implied from $\beta_1^{BE}$; secondly, $B_u < E_u$ is the implicit integrity constraints which always hold for each $\mathcal{CQLC}^t$ query, so the constraints with the form as (2') are also implied from $\beta_1^{BE}$.

Given two adjacent nodes $n_i'$ and $n_{i+1}'$ in the query $Q_2$. If the axis inbetween is \textit{descendant}, we have the following constraints:

$$B_i' < B_{i+1}', \quad B_{i+1}' < E_{i+1}', \quad E_i' < E_{i+1}', \quad L_i' < L_{i+1}'$$

Clearly, given any containment mapping $\rho$, the mapped constraints $\rho(B_i' < B_{i+1}')$ and $\rho(B_{i+1}' < E_{i+1}')$ are always implied from the $\beta_i^{L} \land \beta_i^{BE}$, as we mentioned above. Consequently, the only mapped constraints which are possibly not implied from $\beta_i^{BE} \land \beta_i^{L}$ are $\rho(E_{i+1}' < E_i')$ and $\rho(L_i' < L_{i+1}')$.

Similarly, for the axis \textit{child}, the possible constraints are $\rho(E_{i+1}' < E_i')$ and $\rho(L_{i+1}' + 1 = L_{i+1}')$, with a minor difference on \textit{level} variables.
Following the same argument, for the axis following, we have the constraints:

\[ B'_i < E'_i, E'_i < B'_{i+1}, B'_{i+1} < E'_{i+1} \]

and the only mapped constraint which could possibly not be implied from \( \beta^BE_{i} \land \beta^L_i \) is \( \rho(E'_i < B'_{i+1}) \).

Now consider all the possible containment mappings from the child and descendant axes. Since these two axes distinguish from each other only on the level variables, we call them vertical axes. Correspondingly, horizontal axes means here following axes. There are two possible mappings from any vertical axis between nodes \( n'_i \) and \( n'_{i+1} \).

1. a set of vertical connected axes between nodes \( n_u \) and \( n_v \), where \( u < v \) (cf. Figure 8.11 (a)). Let the mapping \( \rho \) map the node \( n'_i \) to \( n_u \) and \( n'_{i+1} \) to \( n_v \). We have \( \rho(E'_i < E'_{i+1}) = E_u < E_v \). Since there is no horizontal axis between \( n_u \) and \( n_v \), it is obvious that \( E_u < E_v \) can be implied from \( \beta^L_i \land \beta^BE_i \), because the begin variables and end variables between the node \( n_u \) and \( n_v \) are totally ordered with the form \( B_u < B_{u+1} < \ldots < E_{u+1} < E_u \). As a result, if there is a mapped constraint concerning variables from \( n'_i \) and \( n'_{i+1} \), which can not be implied from \( \beta^L_i \land \beta^BE_i \), it has to be \( L_u < L_v \) (or \( L_{u+1} = L_v \)). As a matter of fact, this happens only if the vertical axis occurs to be child, and the number of axes between \( n_u \) and \( n_v \) is more than one.

As a result, in this case, the only possible mapped constraints which can not be implied from the \( \beta^L_i \land \beta^BE_i \) have the form \( L_{u+1} = L_v \).

2. a set of vertical and horizontal (following) axes between nodes \( n_u \) and \( n_v \), where \( u < v \) (cf. Figure 8.11 (b)).

Let the mapping \( \rho \) map the node \( n'_i \) to \( n_u \) and \( n'_{i+1} \) to \( n_v \). We have \( \rho(E'_i < E'_{i+1}) = E_u < E_v \). Since there is at least one horizontal axis between \( n_u \) and \( n_v \), it is possible that \( E_u < E_v \) can not be implied from the \( \beta^L_i \land \beta^BE_i \), because the begin variables and end variables between the node \( n_u \) and \( n_v \) are not anymore totally ordered.

Moreover, the mapped constraint \( L_u < L_v \) (or \( L_{u+1} = L_v \)) can not be implied from \( \beta^L_i \). To see this, consider the example illustrated in Figure 8.11 (b). Without loss of generality, let all the axes of \( Q_i \) be vertical, except for the following one between \( n_j \) and \( n_{j+1} \). Then \( \beta^L_i \) has the form

\[ L_1 < L_2 < \ldots < L_u < \ldots < L_j \land L_{j+1} < L_{j+2} < \ldots < L_v < \ldots < L_n \]

It is obvious that \( \beta^L_i \rightarrow L_u < L_v \).

As a result, if there is a mapped constraint concerning variables from \( n'_i \) and \( n'_{i+1} \) which can not be implied from \( \beta^BE_i \land \beta^L_i \), it has to be \( L_u < L_v \).
8.3. CONTAINMENT CHECKING OF CQLC$^T$

Figure 8.11: Possible mappings from a vertical axis

(or $L_u + 1 = L_v$). Also possible alternative is of course the constraint $E_u < E_v$. However, it suffices if we can definitely decide one (the one with level
variables).

Now we are ready for the selection of $\alpha_i$. Given a containment mapping $\rho_i$. Let $c$ be the atomic constraint in $\beta_2^{BE} \land \beta_2^L$. If $\beta_1^{BE} \land \beta_1^L \rightarrow \rho_i(c)$, we call $c$ the inconsistent atomic constraint with respect to $\rho_i$.

If the inconsistent atomic constraint $c$ is from a vertical axes (child or descendant), then the constraint $L_u < L_v (u < v)$ or $L_u + 1 = L_v (u < v)$ is selected as $\alpha_i$ (because we have shown above that it has to exist); otherwise, if the constraint is from a horizontal axes, then $E_u < B_v (u < v)$ is selected.

Consequently, each $\alpha_i$ has the form from one of the following:

$$\{ E_u < B_v, L_u < L_v, L_u + 1 = L_v \} (u < v)$$

Now it has to be proved that

$$\beta_1^{BE} \land \beta_1^L \rightarrow \alpha_1 \lor \ldots \lor \alpha_u$$

W.l.o.g, let $\alpha_1, \ldots, \alpha_j (1 \leq j \leq u)$ be the constraints with the form of $E_u < B_v$ and $\alpha_{j+1}, \ldots, \alpha_u$ be with the form of $L_u < L_v$ or $L_u + 1 = L_v$. We have to prove that

$$\beta_1^{BE} \rightarrow \alpha_1 \lor \ldots \lor \alpha_j$$

and

$$\beta_1^L \rightarrow \alpha_{j+1} \lor \ldots \lor \alpha_u$$

Given $\alpha_i$ and $\beta_1^{BE}$, it is trivial to prove that $\beta_1^{BE} \rightarrow \alpha_i$ is equivalent to that $\beta_1^{BE} \land \neg \alpha_i$ is satisfiable.

Given $\alpha_i$ with the form $E_u < B_v$, we have $\neg \alpha = E_u \geq B_v$.

Now the task is to prove

$$\beta_1^{BE} \land \neg \alpha_1 \land \ldots \land \neg \alpha_j$$
is satisfiable. Recall that for each \( i(1 \leq i \leq j) \),
\[
\beta_i^{BE} \land \neg \alpha_i
\]
is satisfiable.
The relationship between the variables can be best illustrated with the inequality graphs [Klu88,GSW96] of the constraints. Given a (conjunction) set \( \beta \) of atomic constraints, the inequality graph of \( \beta \), denoted as \( G(\beta) \), is a graph constructed as follows: \(^4\)

- The nodes in the graph are the variables in \( \beta \).
- There is a directed arc from node \( n_1 \) to node \( n_2 \) if the inequality \( n_1 < n_2 \) or \( n_1 \leq n_2 \) is in \( \beta \). Such an arc is labeled as "<" or "\( \leq \)" according to whether the inequality is strict (\( < \)) or nonstrict (\( \leq \)).

The set \( \beta \) is unsatisfiable if and only if there is a directed cycle with at least one edge labeled with \( < \).

Given any \( \beta_i^{BE} \), the inequality graph of it can be obtained according to the above given rules. For instance, Figure 8.12 shows the inequality graph of the query \( a/child ::a/desc ::a/following ::a/child ::a \). This graph concerns only \( B_8 \) and \( E_8 \). It is obvious that there is no directed cycle in this graph.

Since each \( \neg \alpha_i(1 \leq i \leq j) \) has the form \( E_u \geq B_v \), where \( u < v \), we consider all the possible constraints in this form, and for each such a constraint, we add an edge into the equality graph of \( \beta_i^{BE} \). For instance, in Figure 8.13 we add edges for the following constraints: (note that to simplify the presentation, we omit the label \( < \) and \( \leq \) on the edges.)

\[
\begin{align*}
E_1 & \geq B_2 \\
E_1 & \geq B_3 \\
E_1 & \geq B_4 \\
E_1 & \geq B_5 \\
E_2 & \geq B_3 \\
E_2 & \geq B_4 \\
E_2 & \geq B_5 \\
E_4 & \geq B_5
\end{align*}
\]

One might notice that the edges of the constraints \( E_3 \geq B_4 \) and \( E_3 \geq B_5 \) are not added. In fact, these are the only edges that could have led to a directed cycle. This is because the negated form of them (\( E_3 < B_3 \) and \( E_3 < B_5 \) respectively) are implied from \( \beta_i^{BE} \), and as a result, they do not belong to the \( \alpha_i \)’s.

\(^4\)Since there are no constants and equalities involved here, we leave out the rules concerning constants and equalities.
8.3. CONTAINMENT CHECKING OF CQLC\textsuperscript{T}

It is clear that there does not exist any directed cycle in the inequality graph after adding all the possible edges from the constraints with the form \( E_u \geq B_v \), where \( u < v \). Because for each \( E \) node where there are incoming edges from the constraints with the form \( E_u \geq B_v \), there is no outgoing edge (\( E_1, E_2, E_4 \) in our example). Such \( E \) nodes can not contribute to the composition of a cycle. As a result, we have shown that

\[
\beta_1^{BE} \land \neg \alpha_1 \land \ldots \land \neg \alpha_j
\]

remains satisfiable. Correspondingly,

\[
\beta_1^{BE} \not\rightarrow (\alpha_1 \land \ldots \land \alpha_j)
\]

holds.

As far as the constraints over level variables are concerned, we only discuss the form \( L_u < L_v \) where \( u < v \), since the constraints with the form \( L_u + 1 = L_v \) can be argued in the similar way.

We again consider the inequality graph of \( \beta_1^L \). Since level variables are not totally ordered, (due to the following axis), they are only ordered in the partitions. For instance, Figure 8.14 illustrates the inequality graph of constraints over level variables from the query \( \text{a/desc :a/desc :a/following :a/desc :a} \). The partitions are \( \{L_1, L_2, L_3\} \) and \( \{L_4, L_5\} \). Inside such partitions the variables are totally ordered.

Again we add all the possible edges from the atomic constraints with the form \( L_u \geq L_v \), where \( u < v \). Note that each such edge alone should not build a directed cycle. This means that the starting and ending point of each edge should not belong to the same partition. As shown in Figure 8.15, after adding all such edges, there is no directed cycle. As a matter of fact, any directed cycle can only exist within the single partitions, and the correlation among the added edges does not help building any directed cycle.

By this argument, we have proved that

\[
\beta_1^L \not\rightarrow \alpha_{j+1} \lor \ldots \lor \alpha_u
\]

As a result, it is obtained that

\[
\beta_1^{BE} \land \beta_1^L \not\rightarrow (\alpha_1 \lor \ldots \lor \alpha_u)
\]

holds, and accordingly,

\[
\beta_1^{BE} \land \beta_1^L \not\rightarrow \rho_1(\beta_2^{BE} \land \beta_2^L) \lor \ldots \lor \rho_u(\beta_2^{BE} \land \beta_2^L)
\]

holds too. \( \square \)
Figure 8.12: The inequality graph concerning \textit{begin} and \textit{end} variables of the query \\
a/child ::a/desc ::a/following ::a/child ::a.

Figure 8.13: The resulting inequality graph after adding all the possible constraints with the form \( E_u \geq B_v \), where \( u < v \).

\[
\begin{align*}
L_1 & \overset{<}{\rightarrow} L_2 \overset{<}{\rightarrow} L_3 \quad L_4 \overset{<}{\rightarrow} E_5
\end{align*}
\]

Figure 8.14: The inequality graph concerning \textit{level} variables of the query \\
a/desc ::a/desc ::a/following ::a/desc ::a.

Figure 8.15: The resulting inequality graph after adding all the possible constraints with the form \( L_u \geq B_v \), where \( u < v \).
8.3. CONTAINMENT CHECKING OF CQLC<sup>T</sup>

Efficiency of the linear constraints implication test

The constraints in \( \beta^{BE} \) are ordinary arithmetic inequalities of the form \( X < Y \) or \( X > Y \), which have been intensively studied [Ull89]. Given two sets of arithmetic inequalities \( S \) and \( T \), the implication test of \( S \rightarrow T \) takes \( O(n^3) \) time when there are \( n \) inequalities in \( S \) [Ull89].

On the other hand, the constraints in \( \beta^L \) are more complicated, in which arithmetic operations such as \( L_1 + 1 = L_2 \) are involved. Surprisingly, there are no explicitly known results on the efficiency of the implication test on such constraints, although they appear quite frequently in the ordinary SQL queries.

It is observed that the linear constraints of \( \beta^L \) are the fragment of Quantifier-Free Presburger (QFP) formulas, which is a decidable theory that is frequently used in formal verification of both hardware design and software [Str02].

**Definition 26** The Quantifier-Free Presburger (QFP) is the first-order theory over atomic formulas of the form

\[
\sum_{i=1}^{n} a_i x_i \sim c
\]

where \( a_i \) and \( c \) are integer constants, \( x_i \)'s are variables ranging over integers, and \( \sim \) is an operation from \( \{=, \neq, \leq, >, \geq \} \). The semantics of these operators are the usual ones.

A formula \( f \) is either an atomic formula, or is constructed from formulas \( f_1 \) and \( f_2 \) recursively as follows:

\[
f ::= \neg f_1 \mid f_1 \land f_2 \mid f_1 \lor f_2
\]

Now consider the implication test of the linear constraints in \( \beta^L \)'s. The problem we are dealing with can be formulated as follows: given two conjunctions of linear constraints \( p_1 \land p_2 \land \ldots \land p_n \) and \( q_1 \land q_2 \land \ldots \land q_m \), where each constraint is of the form \( L_1+1 = L_2 \) or \( L_1 < L_2 \), we have to show whether

\[
p_1 \land p_2 \land \ldots \land p_n \rightarrow q_1 \land q_2 \land \ldots \land q_m
\]

is valid, which is equivalent to the following formula:

\[
\neg(p_1 \land p_2 \land \ldots \land p_n) \lor (q_1 \land q_2 \land \ldots \land q_m)
\]

If the above formula is valid, then its negated form:

\[
(p_1 \land p_2 \land \ldots \land p_n) \land (\neg q_1 \lor \neg q_2 \lor \ldots \lor \neg q_m)
\]

should be unsatisfiable.
Clearly, the implication problem of the linear constraints in $\beta^L$ can be reduced to the co-problem of satisfiability problem of QFP formulas, which is NP-complete [VGD02].

Now the question arises, is it possible to reduce the bound of the satisfiability problem of QFP, due to the simple form of the constraints in $\beta^L$? The answer is yes, and the algorithm is in polynomial time.

The first exponential explosion of the decision procedure is the so-called "case splitting", i.e. transforming the formula into Disjunctive Normal Form (DNF). The number of clauses in the resulting formula can be exponential in the size of the original formula. Obviously this exponential explosion does not happen on the formulas we are interested in, because they are already in normal form.

After transforming the formula into DNF, we have to solve the satisfiability of the conjunctions of a set of atomic QFP formulas one by one. For instance, given integer variables $x_1, x_2$ and $x_3$, the formula $\varphi$ is such a conjunction:

$$\varphi = x_1-x_2 \leq 0 \land x_1-x_3 \leq 0 \land x_1-2x_2-x_3 \geq 0 \land x_3 \geq 1$$

The most commonly used method is the Fourier-Motzkin (FM) variable elimination method [Str02], which is roughly described as follows. Given a conjunction of $m$ atomic QFP formulas over $n$ integer variables $x_1, x_2, \ldots, x_n$, the FM method eliminates the variables in some given order. Each variable is eliminated by projecting its constraints on the rest of the system. The algorithm halts either a contradiction is obtained — which means the formula is not satisfiable —, or at last one variable is left without contradiction with the answer yes.

**Example 30** Consider the following formula with integer variables $L_1, L_2, L_3$ and $L_4$:

$$L_2-L_1 > 0$$

$$\land L_3-L_2 = 1$$

$$\land L_4-L_3 > 0$$

$$\land L_4-L_1 \leq 0$$

The FM variable elimination method is applied as follows. First the equation $L_3-L_2 = 1$ together with one variable (L_2 or L_3) are eliminated. W.l.o.g. we choose $L_3$. We obtain the following:

$$L_2-L_1 > 0$$

$$\land L_4-L_2 > 1$$

$$\land L_4-L_1 \leq 0$$

Next we eliminate $L_1$. Observe that there are two constraints in the above formula related to $L_1$, from which the upper bound $L_2$ ($L_1 < L_2$) and the lower bound $L_4$
(L₁ ≥ L₄) are obtained. As a result, the new constraint L₂−L₄ ≤ 0 is added, after
the elimination of L₁. Now we have the following formula:

\[ L₄ - L₂ > 1 \]
\[ \land \ L₂ - L₄ \leq 0 \]

Now removing either L₂ or L₄ will result in the constraint 1 ≤ 0. This leads to a
contradiction. As a result, we have proven that

\[ L₁ < L₂ \]
\[ \land \ L₂ + 1 = L₃ \rightarrow L₁ < L₄ \]
\[ \land \ L₃ < L₄ \]

Generally, the FM method can result in the worst case exponential number of
constraints. However, since the variables are connected in such a way, that at most
two variables occur in each formula. This simplicity of syntax does not increase
the number of constraints. For this "Two Variables per Inequality" special case,
the process takes \( O(mn^2) \) worst-case time [Pug91, CM94], where \( m \) is the
number of constraints and \( n \) is the number of variables.

Equipped with the polynomial time upper bound of the implication test of the linear
constraints, we are ready to give the polynomial time algorithm for the containment
checking of \( CQLC^t \) where both acyclicity and homomorphism properties hold.

The polynomial time algorithm

In the following, we show that given \( Q₁ \) and \( Q₂ \) as two \( CQLC^t \) queries with forward
axes, there is a polynomial time algorithm for the containment test \( Q₁ ⊆ Q₂ \). Note
that this polynomial time algorithm is not obvious. Although the homomorphism
property holds, and the implication tests are in polynomial time, there could still
be an exponential number of containment mappings from \( Q₂ \) to \( Q₁ \), even when \( Q₁ \)
and \( Q₂ \) are acyclic. To avoid this exponential explosion, the algorithm does not
generate all such mappings.

Next we give the polynomial time algorithm. The principle is adapted from the
one proposed by Chekuri and Rajaraman. Given two \( CQLC^t \) queries \( Q₁ \) and \( Q₂ \)
with only forward axes as follows:

\[ Q₁ : q \leftarrow e(T₁, B₁', E₁', L₁'), \ldots, e(Tₘ, Bₘ', Eₘ', Lₘ'), I₁. \]
\[ Q₂ : q \leftarrow e(T₁, B₁, E₁, L₁), \ldots, e(T₁, B₁, E₁, L₁'), I₂. \]

Moreover, let \( T \) be the join tree of \( Q₂ \) with nodes \( n₁, n₂, \ldots, nᵢ \), and \( sᵢ \) be the
subgoal \( e(Tᵢ, Bᵢ, Eᵢ, Lᵢ) \) of \( Q₂ \). For each node \( nᵢ \), there is a set of containment
mappings \( Mᵢ \). Let \( Sᵢ \) be the set of subgoals of \( \{e(Tⱼ, Bⱼ, Eⱼ, Lⱼ)\} (1 \leq j \leq i) \) in \( Q₂ \)
and \( I₂^i \) be all the constraints concerning variables on the domain of \( Sᵢ \).
The principle of the algorithm is as follows: each time one takes two adjacent nodes in the join tree and tests the partial mappings from them to $Q_1$. Since the variables are connected in the form of a chain, if every partial mapping is successful, the union of all such partial mappings is the containment mapping from $Q_2$ to $Q_1$. We prove also the algorithm is sound and complete.

**Algorithm HomoContainment**

1. Initialize $M_i(1 \leq i \leq l)$ as follows: For each partial mapping $\rho$ from $Q_2$ to $Q_1$ that maps $s_i$ to some subgoal of $Q_1$, add $\rho$ into $M_i$.

2. Process the nodes from $n_2$ to $n_l$ as follows: Suppose $n_i$ is the node in $T$, such that $n_{i-1}$ has been processed. Let $I_2^{(i-1,i)}$ be the constraints concerning only the variables from nodes $n_{i-1}$ and $n_i$. For each $\rho \in M_i$, if there is a mapping $\varphi \in M_{i-1}$, such that the following holds:

$$I_1 \rightarrow \rho(\varphi(\rho|^{(i-1,i)}))$$

(note that $\rho$ maps only variables in $s_i$ and $\varphi$ maps only variables in $s_{i-1}$; hence the sequence of the application of them is irrelevant) then $\rho$ remains in $M_i$. Otherwise, $\rho$ is removed from $M_i$.

3. $Q_1 \subseteq Q_2$ if and only if $M_l$ is not empty.

**Lemma 11** Algorithm HomoContainment correctly determines whether $Q_1 \subseteq Q_2$.

**Proof 12** We first introduce the projection symbol $\pi$. Let $\phi$ be a containment mapping, $\pi_{s_i}(\phi)$ is the subset of $\phi$ with only the mappings whose domain are the variables in $s_i$.

We use induction on the number of nodes processed, with the following induction hypothesis: after node $n_i$ is processed, the mapping $\rho \in M_i$ if and only if there is a partial mapping $\phi$ from $Q_2$ to $Q_1$ whose domain is the set of subgoals $S_i$, and $I_1 \rightarrow \phi(I_2)$, such that $\pi_{s_i}(\phi) = \rho$. Thus, when the last node $n_l$ has been processed, $M_l$ is not empty if and only if there is a containment mapping $\mu$ from $Q_2$ to $Q_1$, and $I_1 \rightarrow \mu(I_2)$.

- "only if":
  1. **Base**: the induction hypothesis holds for $n_1$, because there is no preceding node for $n_1$ and $M_1$ is initialized in Step 1. Further, there is no constraints concerning only the variables in $s_1$, the set $M_1$ remains after the execution of Step 2.
2. **Induction:** we assume that node \( n_i \) is processed, and there is a mapping \( \rho \) such that \( \rho \in M_i \). According to Step 2, there should be at least one containment mapping \( \varphi \in M_{i-1} \), such that the following holds:

\[
I_1 \rightarrow \rho(\varphi(I_2^{(i-1,i)}))
\]

Since \( \varphi \in M_{i-1} \), according to the induction hypothesis, there is a partial mapping \( \theta \), from \( Q_2 \) to \( Q_1 \) with domain \( S_{i-1} \), such that \( I_1 \rightarrow \theta(I_2^{i-1}) \) and \( \pi_{s_{i-1}}(\theta) = \varphi \).

Thus, the following holds:

\[
I_1 \rightarrow \rho(\varphi(I_2^{(i-1,i)})) \land \theta(I_2^{i-1})
\]

Since \( \varphi \) is the subset of \( \theta \), we obtain:

\[
I_1 \rightarrow \rho(\theta(I_2^{(i-1,i)})) \land \theta(I_2^{i-1})
\]

Clearly, the mappings \( \theta \) and \( \rho \) are consistent, because their domains do not share any common variable. Thus the following holds:

\[
I_1 \rightarrow (\rho \cup \theta)(I_2^i)
\]

Let \( \phi \) be the union of \( \theta \) and \( \rho \). We get

\[
I_1 \rightarrow \phi(I_2^i)
\]

- "if":

1. **Base:** the induction hypothesis holds for \( n_1 \). All the partial mappings from \( e(T_1, B_1, E_1, L_1) \) to \( Q_1 \) are stored in \( M_1 \), and \( M_1 \) remains intact.

2. **Induction:** let \( \phi \) be a partial mapping from \( Q_2 \) to \( Q_1 \) whose domain is the set of subgoals \( S_i \), and \( I_1 \rightarrow \phi(I_2^i) \), such that \( \pi_{s_i}(\phi) = \rho \). By Step 1, \( \rho \) should be in \( M_i \). We have to prove that after Step 2, \( \rho \) still remains in \( M_i \).

   Let \( \theta = \phi \setminus \rho \). Clearly, \( \theta \) is a partial mapping from \( Q_2 \) to \( Q_1 \) whose domain is the set of subgoals \( S_{i-1} \), and \( I_1 \rightarrow \theta(I_2^{i-1}) \). Let \( \varphi = \pi_{s_{i-1}}(\theta) \). Based on the induction hypothesis, \( \varphi \in M_{i-1} \). Now by executing Step 2, \( \rho \) will not be removed, because such a \( \varphi \) exists, and the corresponding implication test also succeeds.

Note that this algorithm can be executed only under the condition that the homomorphism property holds, such that finding all the mappings is not obligatory.

**Example 31** Consider the following XPath queries \( Q_1^{XP} \) and \( Q_2^{XP} \):

- \( Q_1^{XP} \): \( a/child::a/desc::a/following::a/child ::a/child ::a/child ::a \)
- \( Q_2^{XP} \): \( a/desc ::a/following ::a/desc ::a/child ::a \)
The translated CQLC\textsuperscript{t} queries are as follows:

\[ Q_1 : \quad q \leftarrow e(a, B'_1, E'_1, 0), \]
\[ e(a, B'_2, E'_2, L'_2), B'_1 < B'_2, E'_2 < E'_1, 0 + 1 = L'_2, \]
\[ e(a, B'_3, E'_3, L'_3), B'_2 < B'_3, E'_3 < E'_2, B'_3 < E'_3, L'_2 < L'_3, \]
\[ e(a, B'_4, E'_4, L'_4), E'_3 < B'_4, \]
\[ e(a, B'_5, E'_5, L'_5), B'_4 < B'_5, E'_5 < E'_4, L'_4 + 1 = L'_5, \]
\[ e(a, B'_6, E'_6, L'_6), B'_5 < B'_6, E'_6 < E'_5, L'_5 + 1 = L'_6, \]
\[ e(a, B'_7, E'_7, L'_7), B'_6 < B'_7, E'_7 < E'_6, B'_7 < E'_7, L'_6 + 1 = L'_7. \]

\[ Q_2 : \quad q \leftarrow e(a, B_1, E_1, 0), \]
\[ e(a, B_2, E_2, L_2), B_1 < B_2, E_2 < E_1, B_2 < E_2, 0 < L_2, \]
\[ e(a, B_3, E_3, L_3), E_2 < B_3, \]
\[ e(a, B_4, E_4, L_4), B_3 < B_4, E_4 < E_3, L_3 < L_4, \]
\[ e(a, B_5, E_5, L_5), B_4 < B_5, E_5 < E_4, B_5 < E_5, L_4 + 1 = L_5. \]

One might notice that the level variables of the first node of both queries: \( L'_1 \) and \( L_1 \) are replaced with constant 0. This is because XPath requires that the root element of \( Q_2 \) map the root element of \( Q_1 \). In this way, the root element of \( Q_2 \) is forced to map to the that of \( Q_1 \).

To simplify the presentation, we use \( \alpha_i \) to denote the subgoal \( e(T_i, B_i, E_i, L_i) \) in \( Q_2 \), where \( 1 \leq i \leq n \). Similarly, we use \( \alpha'_i \) to denote the subgoal \( e(T'_i, B'_i, E'_i, L'_i) \) in \( Q_1 \), where \( 1 \leq i \leq m \).

Following the Step 1 in the containment checking algorithm, for each subgoal in \( Q_2 \), we initialize the partial mappings with \( M_1, \ldots, M_5 \) as shown in Table 8.1. Since no constraints are involved, every subgoal in \( Q_1 \) can be mapped from \( \alpha_i (2 \leq i \leq 5) \).

In Step 2, we have to process the sets from \( M_1 \) to \( M_5 \) one by one. Since there is no constraint concerning only variables in \( \alpha_1 \), the implication test is skipped. So \( M_1 \) remains intact.

Now consider \( M_2 \). The constraints concerning domain \( S_2 \) are \( B_1 < B_2, E_2 < E_1, B_2 < E_2, 0 < L_2 \). Obviously the mappings \( \alpha_2 \rightarrow \alpha'_2 \) and \( \alpha_2 \rightarrow \alpha'_3 \) are not removed from \( M_2 \) because the corresponding implication tests on constraints hold. As far as \( M_3 \) is concerned, the mappings from \( \alpha_3 \) to \( \alpha'_4, \alpha'_5, \alpha'_6 \) and \( \alpha'_7 \) are all qualified. Because the constraints concerning domain \( S_3 \) is only \( B_2 < E_3 \), and it is easy to check that the corresponding implication tests succeed on all the abovementioned mappings.

The processing on \( M_4 \) and \( M_5 \) follows similarly. Finally there are two mappings in \( M_5 \), which means the containment is successful with the answer "yes".
8.3. CONTAINMENT CHECKING OF CQLC^T

Table 8.1: The mapped subgoals in \( Q_1 \) of Example 31

<table>
<thead>
<tr>
<th>( M_1 )</th>
<th>( M_2 )</th>
<th>( M_3 )</th>
<th>( M_4 )</th>
<th>( M_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha_1 \rightarrow )</td>
<td>( \alpha_2 \rightarrow )</td>
<td>( \alpha_3 \rightarrow )</td>
<td>( \alpha_4 \rightarrow )</td>
<td>( \alpha_5 \rightarrow )</td>
</tr>
</tbody>
</table>

**Step 1**
- \( \{ \alpha'_1 \} \)
- \( \{ \alpha'_1, \alpha'_2, \alpha'_3, \alpha'_4, \alpha'_5, \alpha'_6, \alpha'_7 \} \)
- \( \{ \alpha'_1, \alpha'_2, \alpha'_3, \alpha'_4, \alpha'_5, \alpha'_6, \alpha'_7 \} \)
- \( \{ \alpha'_1, \alpha'_2, \alpha'_3, \alpha'_4, \alpha'_5, \alpha'_6, \alpha'_7 \} \)
- \( \{ \alpha'_1, \alpha'_2, \alpha'_3, \alpha'_4, \alpha'_5, \alpha'_6, \alpha'_7 \} \)

**Step 2**
- \( \{ \alpha'_1 \} \)
- \( \{ \alpha'_2, \alpha'_3 \} \)
- \( \{ \alpha'_4, \alpha'_5, \alpha'_6, \alpha'_7 \} \)
- \( \{ \alpha'_5, \alpha'_6, \alpha'_7 \} \)
- \( \{ \alpha'_6, \alpha'_7 \} \)

**Example 32** Consider the following XPath queries \( Q^{XP}_1 \) and \( Q^{XP}_2 \):

\[
Q^{XP}_1 : \text{a/child ::a/desc ::a}
\]

\[
Q^{XP}_2 : \text{a/desc ::a/child ::a}
\]

The translated CQLC^T queries are as follows:

\[
Q_1 : q \leftarrow e(a, B'_1, E'_1, 0),
\]

\[
e(a, B'_2, E'_2, L'_2), B'_1 < B'_2, E'_2 < E'_1, 0 + 1 = L'_2,
\]

\[
e(a, B'_3, E'_3, L'_3), B'_2 < B'_3, E'_3 < E'_2, B'_3 < E'_3, L'_2 < L'_3.
\]

\[
Q_2 : q \leftarrow e(a, B_1, E_1, 0),
\]

\[
e(a, B_2, E_2, L_2), B_1 < B_2, E_2 < E_1, 0 < L_2,
\]

\[
e(a, B_3, E_3, L_3), B_2 < B_3, E_3 < E_2, B_3 < E_3, L_2 + 1 = L_3.
\]

The initialization in the Step 1 is executed in the similar way as Example 31 (cf. Table 8.2).

Since both \( \alpha'_2 \) and \( \alpha'_3 \) can be mapped from \( \alpha_2 \) (of course the implication test involves the mapping in \( M_1 \)), both of them remain in \( M_2 \). However, this does not exist a mapping from \( \alpha_3 \) to any subgoal in \( Q_1 \), such that the implication test holds. We obtain an empty set in \( M_3 \). As a result, the containment checking fails with the answer "no".

**Theorem 23** (Containment of XPath fragment with forward axes)
Given two CQLC^T queries \( Q_1 \) and \( Q_2 \) with only axes child, descendant and following, the containment test \( Q_1 \subseteq Q_2 \) is in polynomial time.

One may notice that forward axis following-sibling is not included in the above theorem. The reason is that the queries with this axis allowed involve the operation with the backward axis parent.
Table 8.2: The mapped subgoals in $Q_1$ of Example 32

<table>
<thead>
<tr>
<th></th>
<th>$M_1$</th>
<th>$M_2$</th>
<th>$M_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha_1 \rightarrow$</td>
<td>$\alpha_2 \rightarrow$</td>
<td>$\alpha_3 \rightarrow$</td>
</tr>
<tr>
<td>Step 1</td>
<td>${\alpha'_1}$</td>
<td>${\alpha'_1, \alpha'_2, \alpha'_3}$</td>
<td>${\alpha'_1, \alpha'_2, \alpha'_3}$</td>
</tr>
<tr>
<td>Step 2</td>
<td>${\alpha'_1}$</td>
<td>${\alpha'_2, \alpha'_3}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>

8.4 Containment vs. Tree Containment

Different from general query containment, the definition of containment on $CQLC^t$ queries has to take the tree model property of XML data into consideration. Informally, given a relational schema $\mathcal{A}$, all the databases on which the $CQLC^t$ queries are applied are tree databases. The intuition behind a tree database is as follows: (1) it contains a root node, and (2) for each node, there is a unique path from root to it. Next, we give a formal definition.

**Definition 27 (Tree Database)** Given a database schema $\mathcal{A}$, the database $D^t$ is a tree database, iff (1) there is one root tuple $e(t_0, b_0, e_0, 0)$, and (2) for each tuple $e(t_k, b_k, e_k, l_k)$, where $l_k > 1$, there is one and only one set of tuples

$$\{e(t_0, b_0, e_0, 0), e(t_1, b_1, e_1, 1), \ldots, e(t_{k-1}, b_{k-1}, e_{k-1}, l_{k-1})\} \subseteq D^t,$$

such that $b_0, \ldots, b_{k-1}, e_{k-1}, \ldots, e_0$ are totally ordered.

It is obvious that given a schema $\mathcal{A}$, the set of tree databases is a subset of all arbitrary databases from $\mathcal{A}$. Although this property does not affect the query processing of $CQLC^t$, it exerts a considerable influence on containment checking.

**Definition 28 (Tree Containment of $CQLC^t$ queries)** Given two $CQLC^t$ queries $Q_1$ and $Q_2$, $Q_1$ is tree contained in $Q_2$, denoted as $Q_1 \sqsubseteq_t Q_2$, if and only if for all tree databases $D^t$, the answer set of $Q_1(D^t)$ is a subset of $Q_2(D^t)$. $Q_1$ is tree equivalent to $Q_2$, denoted as $Q_1 \equiv_t Q_2$, if both $Q_1 \sqsubseteq_t Q_2$ and $Q_2 \sqsubseteq_t Q_1$.

Clearly, the general containment of two queries is a sufficient condition for the tree containment. The following proposition states this property.

**Proposition 8 (A necessary condition)** Given two $CQLC^t$ queries $Q_1$ and $Q_2$, if $Q_1 \sqsubseteq Q_2$, then $Q_1 \sqsubseteq_t Q_2$.

However, the other direction does not always hold.
8.4. CONTAINMENT VS. TREE CONTAINMENT

\[
\begin{array}{ccc}
\| & a & a \\
* & * & * \\
\| & b & b \\
\end{array}
\]

(a) (b) (c)

Figure 8.16: Queries (a), (b) and (c) are tree equivalent.

**Example 33 (Tree containment)** Consider the following CQLC\(^t\) queries \(Q_1\) and \(Q_2\), (their query patterns are depicted in Figure 8.16(a) and 8.16(b) respectively).

\[
Q_1: \ h() :- \ e(a, B_1, E_1, L_1), e(T, B_2, E_2, L_2), e(b, B_3, E_3, L_3), \\
     B_1 < B_2 < B_3 < E_3 < E_2 < E_1, \\
     L_1 < L_2, L_2+1 = L_3.
\]

\[
Q_2: \ h() :- \ e(a, B_1, E_1, L_1), e(T, B_2, E_2, L_2), e(b, B_3, E_3, L_3), \\
     B_1 < B_2 < B_3 < E_3 < E_2 < E_1, \\
     L_1+1 = L_2, L_2 < L_3. 
\]

It can be shown that \(Q_1 \not\subseteq Q_2\), but \(Q_1 \subseteq_t Q_2\). The intuition behind is that all the counter-examples which may be used to prove that \(Q_1 \not\subseteq Q_2\) are not tree databases.

Example 33 shows that the traditional containment test algorithm does not suffice to check the tree containment of CQLC\(^t\). It may happen that even if the general containment checking fails, the tree containment is still successful. In the rest of this section, we mainly deal with the following question: how to modify the queries (without changing the semantics) to assure that the general containment algorithm can be applied for tree containment?

### 8.4.1 Elimination of Wildcards

It is shown in Example 33 that if both wildcards and descendant edges are allowed in the XPath query, the tree containment is not equivalent to normal containment. Now, the following question arises: can the wildcards be eliminated from the CQLC\(^t\) query, without modifying the semantics?

A wildcard delivers the information about the length of the query path, without specifying the label of the node. The length information is more accurate than the transitive closure on variables in the common sense. For instance, the constraint
8. QUERY EQUIVALENCE OVER TREES

\[ X+2 \leq Y \] delivers more accurate information than the constraint \( X < Y \). To capture this semantics, we have to extend the vocabulary of the linear constraints of CQLC\(^t\) queries with integer constants.

Clearly, proposing a general wildcard elimination algorithm is beyond our scope. Consider that there are 8 standard axes and a wildcard could occur anywhere in the XPath queries, there exists an exponential number of combinations. Nevertheless, at the end of the chapter we enumerate such situations and give a general discussion.

It is observed that most intuitive examples showing that the normal containment is inequivalent to tree containment have both axes \texttt{descendant::} and \texttt{child::} involved. In the following section, we concentrate on the fragment discussed in the previous section, the CQLC\(^t\) queries with only forward axes \texttt{descendant::} and \texttt{child::}. We use the term homo-CQLC\(^t\) for this fragment.

The axis \texttt{following::} in this fragment is left out, because generally in an XPath query, one uses constraints over level variables to represent vertical axes and on the other hand, constraints over the begin and end variables play an important role for the horizontal axes. If the wildcard is between a horizontal axis and a vertical one, it is hard to transfer the constraints which are related to the wildcard into other constraints. Whether one can eliminate such a wildcard and how is still an open problem.

**Definition 29 (Wildcards Elimination)**

Given a homo-CQLC\(^t\) query \( Q \) as follows:

\[
Q : \ h() \ :- \ \text{core}(Q), \beta^{BE}, \beta^{L}.
\]

the corresponding query, after applying the wildcard elimination:

\[
Q_{el} : \ h() \ :- \ \text{core}(Q_{el}), \beta^{BE}_{el}, \beta^{L}_{el}.
\]

can be obtained by the following steps:

1. For each predicate \( e(T, B_i, E_i, L_i) \) in \( \text{core}(Q) \) representing the wildcard, apply Fourier-Motzkin (FM) variable elimination on variables \( B_i, E_i \) and \( L_i \) respectively.

2. Delete all the predicates corresponding to wildcards as well.

**Example 34 (Wildcard elimination)** Given the query \( a/*/b \), the corresponding CQLC\(^t\) is as follows:

\[
Q : \ h() \ :- \ e(a, B_1, E_1, L_1), e(W, B_2, E_2, L_2), e(b, B_3, E_3, L_3), \\
B_1 < B_2 < B_3 < E_3 < E_2 < E_1, \\
L_1+1 = L_2, L_2 < L_3.
\]
8.4. CONTAINMENT VS. TREE CONTAINMENT

Since \((L_1+1 = L_2, L_2 < L_3)\rightarrow L_1+1 < L_3\), and \((B_1 < B_2, B_2 < B_3)\rightarrow B_1 < B_3\), as well as \((E_3 < E_2, E_2 < E_1)\rightarrow E_3 < E_1\), a new query after the wildcard elimination has the form:

\[
Q_{el} : h() := e(a, B_1, E_1, L_1), e(b, B_3, E_3, L_3), \\
B_1 < B_3 < E_3 < E_1, L_1+1 < L_3.
\]

The following lemma shows that any homo-\(\text{CQLC}^t\) query \(Q\) and its corresponding \(Q_{el}\) are tree equivalent.

**Lemma 12** Given a homo-\(\text{CQLC}^t\) query \(Q\) and the corresponding \(Q_{el}\) after wildcard elimination, \(Q \equiv_t Q_{el}\) holds.

**Proof 13** We give the proof by induction. We prove that given any \(\text{CQLC}^t\) query \(Q\), and the new query \(Q_{el}\) by eliminating one wildcard, \(Q \equiv_t Q_{el}\).

There are 4 different types of expressions, in which a wildcard may appear, namely

\[
t_1/ * / t_3, \quad t_1/ * / t_3, \quad t_1/ * / / t_3, \quad t_1/ / / / t_3
\]

where \(t_1\) and \(t_3\) are label constants or variables. In the rest of the proof, we assume that the subgoals for the node \(t_1, * , t_3\) are

\[
e(t_1, B_1, E_1, L_1), e(W, B_2, E_2, L_2), e(t_3, B_3, E_3, L_3)
\]

respectively.

Note also that the proof of one direction, namely \(Q \equiv_t Q_{el}\) is trivial, because \(Q_{el}\) is obtained by simply removing some subgoals from \(Q\). In the rest of the proof we give the proof for the other direction.

Assume that there is a tree database \(D^t\) and a tuple \(t\), such that \(t \in Q_{el}(D^t)\). We have to show that \(t \in Q(D^t)\).

**Case 1:** \(t_1/ * / t_3\). From the translation rules in Section 8.2.2, the constraints \(L_1+1 = L_2, L_2+1 = L_3\) can be obtained. We get \(L_1+2 = L_3\) derived. Similarly, the transitive closure \(B_1 < B_3\) and \(E_3 < E_1\) can be obtained from the constraints \(B_1 < B_2 < B_3 < E_3 < E_2 < E_1\). Let \(e(t_1, b_1, e_1, l_1)\) and \(e(t_3, b_3, e_3, l_3)\) be the instantiation of the subgoals concerning nodes labeled with \(t_1\) and \(t_3\) respectively. From the definition of a tree database, there exists a node \(e(*, b_2, e_2, l_2)\) inbetween, which can be used as a instantiation of the subgoal \(e(W, B_2, E_2, L_2)\).

**Case 2:** \(t_1/ / / t_3\). Similarly, the constraints \(L_1 < L_3, L_2+1 = L_3\) can be obtained, and from which \(L_1+1 < L_3\) is derived. Similar as Case 1, Let \(e(t_1, b_1, e_1, l_1)\) and \(e(t_3, b_3, e_3, l_3)\) be the instantiation of the subgoals concerning nodes labeled \(t_1\) and \(t_3\) respectively. From the definition of a tree database, there exists at least one
node $e(\ast, b_2, e_2, l_2)$ inbetween, such that $l_1 + 1 = l_2$ and $l_2 < l_3$, which can be used as a instantiation of the subgoal $e(W, B_2, E_2, L_2)$.

**Case 3 and Case 4:** similar as Case 2.

After the wildcard elimination, we obtain a tree equivalent query $Q_{el}$. Comparing with $Q$, $Q_{el}$ does not contain the wildcards any more, and the semantics information on the nodes containing wildcards is transformed into linear constraints in $Q_{el}$. The following lemma shows that considering the queries without wildcard, the normal containment is equivalent to tree containment.

**Lemma 13** Given two homo-CQLC$^t$ queries $Q_1$ and $Q_2$ after wildcard elimination, $Q_1 \subseteq Q_2$ iff $Q_{el1} \subseteq Q_{el2}$.

**Proof 14** Only if: trivial. Since $Q_1$ is contained in $Q_2$ over all databases, it is obvious that $Q_1$ is also contained in $Q_2$ for all tree databases.

If: We prove that if $Q_1 \not\subseteq Q_2$, then $Q_{el1} \not\subseteq Q_{el2}$. Assume that there is a database $D$ and a tuple $t$, such that $t \in Q_1(D)$, but $t \notin Q_2(D)$. If $D$ is tree database, then it is fine. Otherwise, we construct a tree database $D'$ from $D$, such that $t \in Q_1(D')$, but $t \notin Q_2(D')$. For each path from root to the leave node, if the level number is not continuous, then fill out inbetween new nodes, whose labels do not occur in $Q_1$ and $Q_2$. We call this new tree database $D'$. It can be obtained that $t \in Q_1(D')$, since $Q \subseteq Q'$. It is also obvious that no new instantiations from variables in $Q_2$ to $D'$ is possible, since the label variables do not match. So if $t \notin Q_2(D)$, it remains that $t \notin Q_2(D')$.

**Theorem 24** Given two homo-CQLC$^t$ queries $Q_1$ and $Q_2$, the tree containment checking problem $Q_1 \subseteq Q_2$ is in polynomial time.

**Proof 15** To solve the tree containment problem, we have to first eliminate all the wildcards in both $Q_1$ and $Q_2$, which takes polynomial time over the size of both queries [Str02, Pug91].

Next, the queries after wildcard elimination $Q_{el1}$ and $Q_{el2}$ remain totally ordered, so that the homomorphism property still holds. As a result, the general containment checking problem of $Q_{el1} \subseteq Q_{el2}$ remains in polynomial time.

### 8.4.2 Wildcard elimination with all standard axes

As far as the wildcards are concerned, there are critical wildcards, non-critical wildcards, and non-removable wildcards. Critical wildcards have the impact on
the containment checking of XPath queries in such a way, that if they are not eliminated, the containment checking would fail, although the tree containment holds. Non-critical wildcards do not have such impact, however they can be eliminated while keeping the tree equivalence of the new query and the old one. As for non-removable wildcards, the elimination of them would result in the loss of information and inequivalence on the modified queries with the original ones. The wildcards in Figure 8.16 are all critical. Of course, it is the position of the wildcards in the path and the adjacent axes that decide whether the wildcards are critical or not. In the following, we consider the wildcards connected with various axes, and analyze the characterization of them.

We consider the fragment of the XPath query with the following form:

\[ X /axis1 :: Y /axis2 :: Z \]

where \( X, Y \) and \( Z \) are possible wildcards. \( axis1 \) and \( axis2 \) are arbitrary axes from the 8 standard ones listed above, in which child, descendant, parent and ancestor are vertical and the rest are horizontal.

Furthermore, we assume that the subgoals for the nodes \( X, Y \) and \( Z \) are

\[ e(X, B_1, E_1, L_1), e(Y, B_2, E_2, L_2), e(Z, B_3, E_3, L_3) \]

respectively.

We start with the vertical axes.

- **Case 1**: \( X /child:: Y /child:: Z \).
  If \( Y \) is a wildcard, by applying axiom (A1), \( L_1 + 2 = L_3 \) is obtained. The transitive closure on begin and end variables, such as \( B_1 < B_3 < E_3 < E_1 \) can be obtained as well (A6). As a result, the wildcard can be removed, but it is non-critical.

- **Case 2**: \( X /descendant:: Y /child:: Z \).
  If \( Y \) is a wildcard, by applying axiom (A2), (A3) and (A4), \( L_1 + 2 \leq L_3 \) is obtained. As a result, the wildcard can be removed too. Note that this wildcard is critical.

- **Case 3**: \( X /child:: Y /descendant:: Z \).
  Similar as Case 2. If \( Y \) is a wildcard, it is critical and has to be removed.

- **Case 4**: \( X /descendant:: Y /descendant:: Z \).
  Similar as Case 2. If \( Y \) is a wildcard, it is critical and has to be removed.

- **Case 5-8**: the mirrored cases of Case 1-4, on parent and ancestor.
8. QUERY EQUIVALENCE OVER TREES

- **Case 9:** \( X /\text{child}: Y /\text{parent}: Z \).
  
  If \( X \) (or \( Z \)) appears to be a wildcard, \( X \) and \( Z \) have to be the same node, which implies that \( X = Z, L_1 = L_3, B_1 = B_3, \) and \( E_1 = E_3 \).
  
  If \( Y \) is a wildcard, and \( X \) and \( Z \) appear to be different constants, the query is inconsistent.

- **Case 10-16:** the other combinations
  
  No wildcard can be removed.

The horizontal axes are somehow different. There is no such transitive closure relationship as shown for \texttt{child} and \texttt{descendant}. As a result, the wildcard elimination is simpler and there is no critical wildcards.

- **Case 1:** \( X /\text{following}: Y /\text{following}: Z \).
  
  If \( Y \) is a wildcard, by applying axiom (A1), \( E_1+2 \leq B_3 \) is obtained. As a result, the wildcard can be removed.

- **Case 2:** \( X /\text{preceding}: Y /\text{preceding}: Z \).
  
  Mirrored case of Case 1.

- **Case 3:** \( X /\text{following-sibling}: Y /\text{following-sibling}: Z \).
  
  If \( Y \) is a wildcard, by applying axiom (A1), \( E_1+2 \leq B_3 \) is obtained, and node \( X \) share the same parent as node \( Z \). As a result, the wildcard can be removed.

- **Case 4:** \( X /\text{preceding}: Y /\text{preceding}: Z \).
  
  Mirrored case of Case 3.

- **Case 5-16:** the other combinations.
  
  No wildcard can be removed.

If one axis from \texttt{axis1} and \texttt{axis2} is vertical (respectively horizontal) and the other is horizontal (respectively vertical), no wildcard can be removed, since the vertical axes are related with \texttt{level} variables, and the horizontal axes are with \texttt{begin} and \texttt{end} variables.

### 8.4.3 Unordered features of XPath

It should be noted that XPath standard has both \texttt{ordered} and \texttt{unordered} features. The axes are ordered, as well as the filters with the form \texttt{[position()=n]}. However, if step expressions are again expressed in filters, it turns out to be unordered. Because the branch can be either on the left or on the right side of the current node. Nevertheless, we consider here the filters with step expressions. Since this feature is also included in the core XPath queries, which is intensively investigated recently.
With each filter in the query, there is a union of two ordered queries which is equivalent to it. As a result, if the query has \( n \) such filters, there are \( 2^n \) ordered queries to represent it.

\[
\begin{array}{c|c}
\text{a} & \text{a} \\
\text{b} & \text{b} \\
\text{c} & \text{c} \\
\text{d} & \text{f} \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{a} & \text{a} \\
\text{b} & \text{b} \\
\text{c} & \text{c} \\
\text{f} & \text{d} \\
\end{array}
\]

(a) \quad (b)

Figure 8.17: Two representations of a sample Core XPath query, which can be distinguished if the ordered query is considered.

**Theorem 25 (Containment of \( CQLC^t \) queries with filter)** The containment checking of XPath queries with filters is in \( \Pi_2^P \). \qed

### 8.5 Conclusion and Open Problems

In this chapter we discussed the query containment problem of XPath queries. Based on the inverted list method of encoding XML data, we first translated the XPath to \( CQLC^t \) queries. Due to the tree model property of XML data, we defined tree containment, which is different from general containment. In order to find the algorithm of tree containment for \( CQLC^t \) queries, we introduced the wildcard elimination algorithm. With this algorithm, a \( CQLC^t \) query can be rewritten to a new query which is tree equivalent to the old one.

Our results show clearly how conventional methods, taken from the rich set of tools developed in the context of relational database, can be successfully utilized in the world of trees. In addition, definitions like tree containment and tree model property give rise to many further investigations concerning the properties that are typical for XML data.

As mentioned before, many problems concerning wildcard elimination remain open. To the best of our knowledge, there is no thorough investigation on this topic. Recently, Chan and Fan [CFZ04] proposed some algorithms concerning wildcard elimination over vertical axes. However, these algorithms are heuristic and the sound and complete proof is missing.
Part IV

Conclusion
9 CONCLUSION

9.1 Main Contributions

In this work we studied query containment problem under various conditions. First we addressed the containment checking problem of some conjunctive query extensions. We solved the following problems:

- We introduced a new method for testing the query containment problem for the conjunctive queries with safe negated subgoals. In comparison to the existing one, the algorithm is much more efficient in the average cases. We have also shown that this algorithm can be naturally extended to the containment checking of unions of CQ's. We have discussed the problem of answering queries using views when both queries and views have negated subgoals. Some motivating examples are given to show the cases which might not be encountered with pure conjunctive queries.

- On the basis of the first result, we addressed the query containment problem for the acyclic conjunctive queries with safe negated subgoals. We showed that the problem is Co-NP complete, by giving both the upper and lower bounds. The proof shows that under the universe \{0,1\}, the containment checking is Co-NP hard. The hardness proof for the containment checking problem under any universe is left open.

- To represent the incomplete information, we introduced disjunctive referential integrity constraints and gave a sound and complete algorithm for checking the containment of conjunctive queries under disjunctive referential and implication constraints. The technique for handling disjunctive referential constraints is related to the well known minimal model semantics for disjunctive logic programming [LMR92]. Our work generalizes the results of Zhang et al. [ZÖ97] in which only referential constraints without disjunctions are considered; we proposed a solution to a question left open so far.

In the second part of this work, we concentrated on the problem of queries (such as XPath) on finite ordered databases (such as XML). We addressed the expressiveness
of XPath queries on both finite words and finite ordered trees. The containment checking problems were discussed as well.

We were confronted with several new challenges on databases and query language. For instance, the databases are ordered and acyclic, which is contrary to the unordered cases in traditional RDBs. Queries are also acyclic, as far as XPath is concerned.

We made the following contributions in this part of the thesis:

- We started with a simple form of finite ordered structure, namely finite words. We analyzed the XPath queries over finite words and proposed a transformation from the fragment of XP(//,* ) to star-free regular expressions. We further showed that the containment problem for this fragment is tractable, by providing a polynomial time algorithm, due to acyclicity and the homomorphism property of the queries.

- In the last chapter of the work we investigated the containment checking problems for various fragments of XPath over finite ordered trees. Based on the "inverted list" encoding method on XML data, we proposed the polynomial transformation of the XPath queries into conjunctive queries with linear constraints. Similar as with words, the containment of XPath fragment with forward axes is tractable. However, if the backward axes are added, the containment problem is in Co-NP.

At last, we considered the tree containment problem. Because the underlying databases are ordered trees, instead of arbitrary structures, There are situations, in which the general containment fails while the tree containment holds. We proposed the wildcard elimination algorithm, to remove some critical wildcards, so that the general containment algorithm can be applied afterwards.

### 9.2 Discussion

We have shown in this thesis that the traditional relational database theory can still be exploited dealing with Internet databases. However, several new features have to be re-considered under this scenario.

We notice that the queries over tree-like structures are mostly acyclic, and some of them have homomorphism properties. Correspondingly, the containment checking complexity results are different. Figure 9.1 illustrates the relationship between them and the corresponding containment checking complexity results. There are several interesting extensions have to be investigated. For instance, the complexity
of query containment $Q_1 \subseteq Q_2$ where $Q_1$ is acyclic and the variables of $Q_2$ are totally ordered, is unknown.

Similar questions can also be raised on query evaluation. For instance, what is the complexity of applying a (cyclic) conjunctive query to a word structure? Recently, some results are proposed for the query evaluation of CQs over acyclic databases [GKS04].

Figure 9.1: The relationship between fragments of CQs and their containment checking

Over both trees and words, we attempted to capture the tree model property (respectively, word model property). Over words, we proposed the normal form for each XP(//,*) expression, by compressing the $A$s and $A*$s, so that the general containment checking can be applied afterwards. Since the fragment is simple (with only two axes) and the word model property is also first-order expressible [EF95], it can be shown that after the normal form rewriting, the general containment is equivalent to word containment. On the other hand, the tree model property is much more complicated. We applied wildcard elimination, by exhaustively enumerating the possible combinations of wildcards and the axes in XPath. As a result, we can only prove the soundness, but not the completeness, which is expected to be done in the future work.
9. CONCLUSION
A RESULTS ON MODEL CHECKING

A.1 Logics in Automated Verification

A.1.1 Kripke Structures

A *Kripke structure* or *transition system* $M$ consists of

- a (finite or infinite) universe $S$ whose elements called *states*;
- a (finite) collection of unary predicates $p, q, \ldots$ which are called *atomic propositions*; and
- a (finite) collection of binary *transition relations* $R$.

The interpretation $\pi(p)$ gives the set of states in which $p$ holds. Intuitively, a Kripke structure is a labeled, directed graph. For instance, in Figure A.1, there are set of states $S = \{u, v, w\}$, atomic propositions $\phi = \{p, q\}$,

$$\pi(p) = \{u, w\}, \pi(q) = \{u\}$$

Finally, the binary relations are $R = \{(u, w), (v, w), (u, v), (v, v)\}$.

![Kripke Structure](image)

Figure A.1: The Kripke Structure $M$

A formula $\varphi$ which is true at a given state $s$ in a structure $M$, is written as

$$(M, s) \models \varphi$$

This problem is known as the *model checking problem*, which is similar to the *query evaluation problem* in database terms. One minor difference is that with model checking one obtains *yes* or *no* as result, while normally a set of tuples are
returned as a database query result. However, it is not difficult to transform any conjunctive query into a boolean query. It should also be noted that in the database point of view, the complexity of model checking problem is considered as combined complexity, while not the data complexity (cf. [Var82]).

A formula $\phi$ is satisfiable in a Kripke structure $M$ if $(M, u) \models \phi$ for some state $u$ of $M$. We say that $\phi$ is satisfiable if it is satisfiable in some Kripke structure.

Among the set of temporal logics, we introduce the well-studied ones, namely the Proposition Linear Temporal Logic (PLTL), Computational Tree Logic (CTL), and CTL* in the following sections. The choice is justified by several well-known results on the equivalence of expressive power. For instance, the equivalence of PLTL and star-free regular expressions on words.

## A.1.2 PLTL

In this section, we give a definition of Propositional Linear Temporal Logic (PLTL). The basic temporal operators of this system are $Fp$ ("sometime p"), $Gp$ ("always p"), $Xp$ ("nexttime p"), and $pUq$ ("p until q"). The formulae of this system are built up from atomic propositions, the truth-function connectives ($\land, \lor, \neg$, etc.) and the above-mentioned temporal operators.

### Semantics

The semantics of a formula $\varphi$ of PLTL with respect to a Kripke structure $M$ is defined as follows. If $M$ is understood, we write $s_0 \models \varphi$, instead of $(M, s_0) \models \varphi$.

1. $s_0 \models p$ iff $s_0 \in \pi(p)$, for atomic proposition $p$
2. $s_0 \models p \land q$ iff $s_0 \models p$ and $s_0 \models q$
3. $s_0 \models \neg p$ iff $s_0 \not\models p$
4. $s_0 \models (pUq)$ iff $\exists j(s_j \models q \land \forall k < j(s_k \models p \land (s_k, s_{k+1}) \in R))$
5. $s_0 \models Xp$ iff $s_1 \models p$ and $(s_0, s_1) \in R$

### Model checking and satisfiability

**Theorem 26 (Complexity of satisfiability for PLTL [SC85])** The problem of testing satisfiability for PLTL is PSPACE-complete.

**Theorem 27 (Complexity of model checking for PLTL [SC85])** The model checking problem for PLTL is PSPACE-complete.
A.1. LOGICS IN AUTOMATED VERIFICATION

A.1.3 CTL and CTL*

In branching time temporal logic, the underlying structure of time is assumed to have a branching tree-like nature.

CTL (Computational Tree Logic) allows basic temporal operators of the form: a
path quantifier – either $A$ ("for all futures") or $E$ ("for some future") – followed by
a single one of the usual linear temporal operators $G$ ("always"), $F$ ("sometime"),
$X$ ("nexttime") or $U$ ("until").

Syntax

In the following we define the syntax of CTL, in which the so called state formulae
and path formulae are involved.

S1 Each atomic proposition $p$ is a state formula
S2 If $p, q$ are state formulae, then so are $p \land q, \neg p$
S3 If $p$ is a path formula then $Ep, Ap$ are state formulae
P0 If $p, q$ are state formulae, then $Xp, pUq$ are path formulae

Intuitively, the syntax of CTL requires stick the path quantifiers $A$ or $E$ to temporal
operators $X$ or $U$ together, so that the formulae appear to be the combination
of $AX$, $AU$, $EX$ and $EU$ subformulae. As a result, neither the path quantifier
followed by a another path quantifier, nor a temporal operator followed by another
temporal operator, is allowed.

Semantics

1. $s_0 \models p$ iff $s_0 \in \pi(p)$, for atomic proposition $p$
2. $s_0 \models p \land q$ iff $s_0 \models p$ and $s_0 \models q$
   $s_0 \models \neg p$ iff $s_0 \not\models p$
3. $s_0 \models EXp$ iff $s_1 \models p$ for some $s_1$ such that $(s_0, s_1) \in R$
   $s_0 \models AXp$ iff $s_1 \models p$ for all $s_1$ such that $(s_0, s_1) \in R$
4. $s_0 \models E(pUq)$ iff there exists a path $s_0, s_1, \ldots$ and some $i \geq 0$, such that $s_i \models q$
   and for all $j$, where $0 \leq j < i$, we have $s_j \models p$
5. $s_0 \models A(pUq)$ iff for all paths $s_0, s_1, \ldots$ and some $i \geq 0$, such that $s_i \models q$ and for
   all $j$, where $0 \leq j < i$, we have $s_j \models p$

PLTL and CTL are incomparable on the expressiveness. For instance, the CTL
expression $EFAGp$ is not expressible in PLTL, and the PLTL expression $FGp$ is
not expressible in CTL. Nevertheless, the simplicity of CTL results in a surprisingly low complexity of model checking problem.

Model checking and satisfiability

Theorem 28 (Complexity of model checking for CTL [Eme90]) The model checking problem for CTL is in deterministic polynomial time.

Theorem 29 (Complexity of satisfiability for CTL [Eme90]) The problem of testing satisfiability for CTL is EXPTIME-complete.

The polynomial time results of model checking of CTL can be explained as follows: the branching-time formulae are interpreted over states of a structure, rather than over execution sequences, and determining truth in a particular structure is much easier and, in many cases in polynomial time. The model checking algorithm of CTL, according to Clarke et al. [CES86], involves a bottom-up labeling method on the tree representation of a CTL formula. This reminds us of the bottom-up query evaluation algorithm of acyclic conjunctive queries on relational databases. As a matter of fact, these two properties are indeed correlated with each other due to the result of Flum et al. [FFG02], which will be elaborated in the further sections.

As we have seen, the syntactic restrictions of CTL significantly limit its expressive power. Therefore the language CTL* is considered. CTL* extends CTL by allowing basic temporal operators where the path quantifiers (A or E) is followed by an arbitrary linear time formula, allowing boolean combinations and nesting, over F, G, X and U. It subsumes both CTL and PLTL.

Theorem 30 (Complexity of model checking for CTL* [Eme90]) The model checking problem for CTL* is PSPACE complete.

Theorem 31 (Complexity of satisfiability for CTL* [Eme90]) The problem of testing satisfiability for CTL* is 2EXPTIME-complete.

A.2 Tree Model Property

As the most important data structure in computer science, tree is also applied in relational database systems. For instance, B+ trees and kd trees are deployed for the data storage to optimize the retrieving of data items. However, only recently, considering trees as database structures, is receiving much attention within the database community, with the tree-like databases as XML and LDAP appearing to be data interchange standard on Internet.
A.2. TREE MODEL PROPERTY

Within the finite model theory and automated verification communities, tree structures have received extensive investigation for several decades. Many interesting results concerning tree structures have been published to explain the decidability of logics and efficiency of model checking algorithms. The following selected results will give an intuition on this broadly researched topic.

A.2.1 Decidability of SnS

It is well-known that first-order logic is not decidable, because it can encode infinite large finite grids, and a run of a Turing Machine can be described by a colored two-dimensional grid, where the configuration extends to the right and time extends downwards [BGG97].

Second order logic, SO, is an extension of first-order logic which allows to quantify over relations. A more interesting fragment of SO is the monadic second order (MSO) logic, which only unary relation variables (set variables) are allowed.

The logic SnS is a monadic second order logic about n-ary tree structures. In SnS we view tree structures as relational structures: the nodes of the tree are the elements of the domain and the propositional constants are viewed as unary predicates. We also view the propositional variables as unary predicates. The binary predicates $S_0, \ldots, S_{n-1}$, where $S_i(x, y)$ means that $y$ is the $i$th child of $x$. The atomic formulas are either of the form $x = y$, $P(x)$, where $P$ is a unary predicate, or of the form $S_i(x, y)$. In addition to Boolean connectives and first-order quantifiers, we allow also monadic second-order quantifiers such as $\exists P$, where $P$ is variable predicate.

An SnS formula is closed if all its individual variables and all its variable predicates are quantified.

**Proposition 9 (Decidability of SnS [Rab69])** The validity problem for closed SnS formulas is decidable. \hfill \Box

For instance, the formula

$$\bigvee_{i=1}^{n} R_i(x, y)$$

means that $y$ is a successor of $x$, which can be denoted as $\text{succ}(x, y)$.

The formula

$$\forall x \exists y (P(x) \rightarrow \text{succ}(x, y) \land P(y))$$

means that every node that satisfies $P$ has a successor that satisfies $P$, or $P$ is downward closed.
The proof in [Rab69] involved a reduction of SnS to tree automata, which is known to be decidable. This idea inspired several later proofs in model checking and XML databases. For instance, the equivalence of Monadic Datalog and MSO on trees was proven by the reduction of Monadic Datalog to tree automata.

### A.2.2 Bisimulation

When the **expressive power** of first-order logic (or relational calculus) is concerned, it is well-known that if two structures $A$ and $B$ are isomorphic to each other, then they can not be distinguished by any first order-logic sentence [EF95]. This means, there does not exist a formula $\varphi$, such that $A \models \varphi$, but $B \not\models \varphi$, or vice versa. The famous Ehrenfeucht-Fraïssé game, is used to prove that certain queries like **even** or transitive closure are not expressible in first-order logic.

However, considering only isomorphism or elementary equivalence \(^1\) under transition systems is often too strict, since differing graphs can exhibit the same possible behaviors, e.g. in form of labeled paths. The most important role is played by the so-called **bisimulation equivalence**. Bisimulation between transition systems not only captures the natural notion of behavioral equivalence between processes, but also corresponds to the nature notion of Ehrenfeucht-Fraïssé equivalence associated with modal quantification.

Consider two Kripke structures $M_1(S_1, \pi_1, R_1)$ and $M_2(S_2, \pi_2, R_2)$. Assume $u \in S_1$ and $v \in S_2$. A binary relation $\sim \subseteq S_1 \times S_2$ is a **bisimilarity** relation if the following holds for each pair $u, v$ of nodes such that $(u, v) \in \sim$:

- $\{p \in \Phi | u \in \pi_1(p)\} = \{p \in \Phi | v \in \pi_2(p)\}$
- If $(u, u') \in R_1$, then there is some $v' \in S_2$ such that $(v, v') \in R_2$ and $u' \sim v'$.
- If $(v, v') \in R_2$, then there is some $u' \in S_1$ such that $(u, u') \in R_1$ and $u' \sim v'$.

Two nodes $u, v$ are **bisimilar**, denoted as $u \sim v$, if they are related by some bisimilarity relation.

For example, the two Kripke structures in Figure A.2 are bisimilarly equivalent. However, the two structures in Figure A.3 are not bisimilar. Firstly, $(u_1, v_1)$ has to be in the relation $\sim$. It follows that $(u_2, v_2)$ and $(u_2, v'_2)$ have to be included in $\sim$ too. Now consider the node pair $(u_2, v_2)$, because $(u_2, u_4) \in R_2$, according to the bisimilarity definition, it is expected that there is a node $v$ in $S_2$, such that $(v, v) \in R_2$ and $u_4 \sim v$, which is unfortunately not the case in the example.

---

\(^1\)they are used interchangeable under finite structures
A.2. TREE MODEL PROPERTY

There are interesting results stating that PLTL, CTL, CTL* can not distinguish the bisimilar structures. In fact, the modal logic (ML) is exactly the bisimulation-invariant fragment of first-order logic.

Theorem 32 (Bisimulation invariance of modal logic [vB83]) The properties definable in ML are precisely the properties that are first-order definable and invariant under bisimulation.

One interesting question concerning the database queries on Kripke structures would be, are there any syntax characterization on conjunctive queries, which can not distinguish the bisimilar structures? The answer is yes, and the acyclic conjunctive queries belong to this fragment. This means, there is no acyclic CQ which can distinguish the structures in Figure A.2. One may notice that assume the variable X is assigned for node $v_3$, and Y for the node $v'_3$ in the right-hand side structure in Figure A.2, the extra built-in predicate $X \neq Y$ would be enough to distinguish the two structures. However there is no contradiction, since the built-in predicate results in a cyclic hypergraph.
A.2.3 Tree Model Property of Temporal Logics

It has been shown in the former sections that the satisfiability of temporal logics is robustly decidable and several model checking algorithms have surprisingly low complexity, such as the linear time algorithm for model checking of CTL. It is obvious that most temporal logic can express properties which are not first-order expressible, such as transitive closure. It is known that the satisfiability of first-order logic is already undecidable.

To unravel this riddle, Vardi pointed out in his seminal paper [Var96] that it was the tree model property of the temporal logics that makes the satisfiability so robustly decidable. The tree model property comes, in turn, from the bisimulation invariance property, which can be stated as follows:

**Proposition 10 (Bisimulation invariance [HM85])** Let $M_1(S_1, \pi_1, R_1)$ and $M_2(S_2, \pi_2, R_2)$ be Kripke structures, and assume $u \in S_1$ and $v \in S_2$. If $u \sim v$, then $(M_1, u) \models \phi$ iff $(M_2, v) \models \phi$, for each modal fixpoint formula $\phi$. □

It is not difficult to show that every Kripke structure $M$ can be "unwound" to a tree structure $M'$, which is bisimilar to $M$. As illustrated in Figure A.4, the Kripke structure $M'$ is obtained from the Kripke structure $M$ in Figure A.1 by "unwinding" the node $v$. It is obvious that $(M, u) \sim (M', u)$.

![Figure A.4: The Kripke Structure $M'$](image)

We do not intend to deeply dig into details of the algorithm for unwinding the Kripke structure to a tree. Interested readers can find it in several literatures [Var96, Hir02].

**Proposition 11 (Tree model property)** [Var96] Let $M = (S, \pi, R)$ be a Kripke structure and assume $u \in S$. Then there is a tree structure $M' = (T, \pi', R')$ such that $u \sim \epsilon$, where $\epsilon$ is the root of the tree. □

A.2.4 Guarded Logics and Acyclic CQs

Tree model property explains partly the decidability of temporal logics. Andréka, van Benthem and Németi [Grä99] suggested that the real reason for the good prop-
erties of modal logics is the so-called guardedness. They named this fragment guarded fragment, or GF of first-order logic.

**Definition 30 (Guarded FO)** The definition of GF is as follows:

1. Every relational atomic formula $R_{x_1} \ldots x_{im}$ or $x_i = x_j$ belongs to GF.
2. GF is closed under boolean operations.
3. if $\bar{x}, \bar{y}$ are tuples of variables, $\alpha(\bar{x}, \bar{y})$ is a positive atomic formula and $\psi(\bar{x}, \bar{y})$ is a formula in GF such that all free variables of $\psi$ occur in $\alpha$, then also the formulae

$$\exists \bar{y}(\alpha(\bar{x}, \bar{y}) \land \psi(\bar{x}, \bar{y}))$$

and

$$\forall \bar{y}(\alpha(\bar{x}, \bar{y}) \rightarrow \psi(\bar{x}, \bar{y}))$$

belongs to GF.

The atom $\alpha(\bar{x}, \bar{y})$ is called the **guard** of the quantifier.

The importance of guarded fragment of logics is that the restrictions of the temporal logics, such as the limitation to unary and binary predicates and the restriction to two variables, can be all removed. As a result, the guarded fragment is a generalized form of the temporal logic.

There are more extensions of GF with fixpoint, which is beyond the scope of this thesis. Interested readers can find more details in the excellent survey from Grädel [Grä99], which follows the one from Vardi [Var96].

On the database point of view, it is known that a relational query is first-order definable, if and only if it is definable in nonrecursive stratified datalog (a.k.a. relational algebra, a.k.a. relation calculus). One natural question would be whether there is a syntax restriction to such queries, so that it is equivalent to the guarded fragment of first-order logic. The interesting results from Flum et al. [FFG02] state that every GF-sentence is equivalent to a boolean acyclic NRSD (short for NonRecursive Stratified Datalog) program and vice versa.

The importance of the result of Flum et al. is that it builds up a nice connection between the well-known acyclic fragment of conjunctive queries and the relatively new results on guarded logics. The relation can be easily understood by a join tree and the corresponding first-order formula as illustrated in Figure A.5.

**Example 35 (Guarded Logic and Acyclic CQ)** Consider the following CQ:

$$q \leftarrow p(X, Y, Z), q(X, Y, T), r(Y, Z, U), s(T, W).$$
The query can be written as the following first-order formula:

\[
\exists X, Y, Z. (p(X, Y, Z) \land \exists T. (q(X, Y, T) \land \exists W. s(T, W)) \land \exists U. (r(Y, Z, U))
\]

where \(p(X, Y, Z)\) (which is also the root of the join tree in Figure A.5) is the guard for both \(q(X, Y, T)\) and \(r(Y, Z, U)\), and in which the subformula \(q(X, Y, T)\) is again the guard of the subformula \(\exists W. s(T, W)\).

From the Example 35 it is shown that the join tree of each acyclic conjunctive query represents exactly the guardedness: every node in the tree which has children is the guard for the subformula represented as child nodes.
B MONADIC DATALOG AND
REGULAR EXPRESSIONS

It is well-known that MSO on trees, denoted as SnS, is decidable. On finite words, the equivalence was also shown between finite automata and MSO.

**Theorem 33 (Equivalence of MSO and finite automata on words)** A language of finite words is recognizable by a finite automaton iff it is definable in monadic second-order logic.

Recently, Gottlob and Koch [GK02a, GK02b] proposed a fragment of datalog: Monadic Datalog, short M-datalog and proved that over trees, it has the same expressiveness with MSO and tree automata. In the next section, we show that over words, which are special form of trees, this fragment has also exactly the same expressiveness as MSO and regular languages.

A datalog program is monadic, if all its IDB predicates are unary. In this section, we concentrate on the Monadic Datalog on words. According to [GS03], Monadic Datalog has the following normal form:

**Definition 31 (Monadic Datalog normal form)** A Monadic Datalog program \( \mathcal{P} \) on words over alphabet \( A \) is in normal form iff each rule is of one of the following forms:

\[
X(x) \quad ::= 
\begin{align*}
& P(x), \\
& X'(x), X''(x), \\
& S(x,y), X'(y), \\
& X'(x), S(x,y).
\end{align*}
\]

where \( X, X', X'' \in \text{IDB}(\mathcal{P}) \), \( S \) is the successor predicate as defined, and \( P \in \{(P_a)_{a \in A}\} \cup \{(-P_a)_{a \in A}\} \).

Next we prove that Monadic Datalog and regular expressions have exactly the same expressiveness on words. The main theorem is stated as follows:

**Theorem 34 (Equivalence of Monadic Datalog and finite automata)** A language of finite words is recognizable by a finite automaton iff it is definable in Monadic Datalog.
We first give the proof for one direction: for each regular expression, there is an equivalent Monadic Datalog program.

**Proposition 12 (Equivalence of Monadic Datalog and finite automata)**
Any regular language is definable in Monadic Datalog.

**Proof 16** We prove by induction on the nesting depth of the recursion + in the regular expression $r$ that there is a sentence $\varphi_r$ in M-datalog normal form, defining the language denoted by $r$.

Note that if it is clear from the context, the concatenation symbol "," will be left out to simplify the presentation. e.g., we write the concatenation as $a b c \ldots$, instead of $a \cdot b \cdot c \cdot \ldots$

**Step 0: Nesting depth is zero** The regular expression $r$ can be easily rewritten to the normal form: $r = r_1 \cup r_2 \cup \ldots \cup r_n$

where every $r_i (1 \leq i \leq n)$ is the concatenation of letters. The M-datalog program $\varphi_r$ has the following form:

$$
\varphi_r(x) \coloneqq \varphi_{r_1}(x).
$$

$$
\ldots
$$

$$
\varphi_r(x) \coloneqq \varphi_{r_n}(x).
$$

For each $r_i (1 \leq i \leq n)$, we assume that it has the form of $a_1 \ldots a_m$, where $a_1, \ldots, a_m$ are letters from alphabet. The following is the M-datalog program for the expression $r_i$:

$$
\varphi_{a_m}(x) \coloneqq P_{a_m}(x).
$$

$$
Q_{a_m}(x) \coloneqq S(x, y), \varphi_{a_m}(y).
$$

$$
\varphi_{a_{m-1} a_m}(x) \coloneqq P_{a_{m-1}}(x), Q_{a_m}(x).
$$

$$
\ldots
$$

$$
\varphi_{a_1 \ldots a_m}(x) \coloneqq P_{a_1}(x), Q_{a_2 \ldots a_m}(x).
$$

Before we move to the next step, we show how the transformation of the concatenation of two sub-expression $r$ and $s$ can be made, where $r$ has zero recursion nesting depth and there is no restriction on $s$. Let $\varphi_s(x)$ be the M-datalog program of $s$, we define $Q_s(x)$ as follows: (Note that in the rest of the proof, the predicate $Q$ will be uniformly used in the same manner.)

$$
Q_s(x) \coloneqq S(x, y), \varphi_s(y).
$$

Let the disjunctive normal form of $r$ is same as above. The M-datalog program $\varphi_{rs}$ has the following form:
\[
\varphi_{r_1 s}(x) \quad : \quad \varphi_{r_1 s}(x).
\]

\[
\vdots
\]

\[
\varphi_{r_n s}(x) \quad : \quad \varphi_{r_n s}(x).
\]

For each \( r_i s (1 \leq i \leq n) \), we simply append \( Q_s \) at the end of the expression \( r_i \).

Again, let us assume that \( r_i \) has form \( a_1 \ldots a_m \), the following is the M-datalog program of \( r_i s \).

\[
\begin{align*}
\varphi_{a_m s}(x) & \quad : \quad P_{a_m}(x), Q_s(x). \\
Q_{a_m s}(x) & \quad : \quad S(x, y), \varphi_{a_m s}(y). \\
\varphi_{a_{m-1} a_m s}(x) & \quad : \quad P_{a_{m-1}}(x), Q_{a_m s}(x). \\
\vdots
\end{align*}
\]

\[
\begin{align*}
\varphi_{a_1 \ldots a_m s}(x) & \quad : \quad P_{a_1}(x), Q_{a_2 \ldots a_m s}(x).
\end{align*}
\]

**Step 1: Nesting depth is one** At first, we introduce the transformation of expression with the form \( r^+ \), where \( r \) has zero recursion nesting depth:

**Base** \( \varphi_{r^+}(x) \quad : \quad \varphi_{r}(x). \)

**Recursion** \( \varphi_{r^+}(x) \quad : \quad \varphi_{r^+}(x). \)

Since \( r \) has zero nesting depth, the transformation of \( \varphi_{r^+}(x) \) is given in the last step.

Similarly, we can rewrite any regular expression into the normal form, \( r = r_1 \cup r_2 \cup \ldots \cup r_n \) where each \( r_i (1 \leq i \leq n) \) is the concatenation of sub-expressions \( s_1, \ldots, s_m \), in which each \( s_i (1 \leq i \leq m) \) has one or zero recursion nesting depth.

Now we are ready to *stitch* the sub-expressions \( s_1, \ldots, s_m \) one by one, starting from \( s_m \). Consider any sub-expression \( s_i (1 \leq i \leq m) \), and another expression \( s \) with the given \( Q_s(x) \). We need to concatenate \( s_i \) and \( s \). Note that \( s_i \) can be either a single letter \( b \), or with recursion \( t^+ \), where \( t \) has zero nesting depth of recursion. The former case is trivial:

\[
\varphi_{bs}(x) \quad : \quad P_b(x), Q_s(x).
\]

For the latter case, we have to append \( Q_s \) in the *Base* part of \( t^+ \):

**Base** \( \varphi_{t^+s}(x) \quad : \quad \varphi_t(x), Q_s(x). \)

**Recursion** \( \varphi_{t^+s}(x) \quad : \quad \varphi_{t^+s}(x). \)

Again, we can also show that the transformation of the concatenation of two sub-expression \( r \) and \( s \) can be made, where \( r \) has one recursion nesting depth and there is no restriction on \( s \) (cf. Step 0).

**Step 2: Induction** Assume that any regular expression which has \( n-1 \) recursion nesting depth, can be transformed to an M-datalog program, we prove that
any regular expression \( r \), which has \( n \) recursion nesting depth, can also be transformed into an M-datalog program.

Given \( r \), whose recursion nesting depth is not more than \( n-1 \), the expression \( r^+ \) can be formulated as follows:

\[
\begin{align*}
\text{Base} & \quad \varphi_{r^+}(x) \leftarrow \varphi_r(x). \\
\text{Recursion} & \quad \varphi_{r^+}(x) \leftarrow \varphi_{r^+}(x).
\end{align*}
\]

Similarly, we can rewrite any regular expression \( r \) into the normal form, \( r = r_1 \cup r_2 \cup \ldots \cup r_n \) where each \( r_i(1 \leq i \leq n) \) is the concatenation of sub-expressions \( s_1, \ldots, s_m \), in which each \( s_i(1 \leq i \leq m) \) has maximum \( n-1 \) recursion nesting depth.

Now we are ready to stitch the sub-expressions \( s_1, \ldots, s_m \) one by one, starting from \( s_m \). Consider any sub-expression \( s_i(1 \leq i \leq m) \), and another expression \( s \) with the given \( Q_s(x) \). We need to concatenate \( s_i \) and \( s \). Note that \( s_i \) can be either a single letter \( (b) \), or with recursion \( (t^+) \), where \( t \) has maximum \( n-1 \) nesting depth of recursion. The former case is trivial:

\[
\varphi_{bs}(x) \leftarrow P_b(x), Q_s(x).
\]

For the latter case, we have to append \( Q_s \) in the Base part of \( t^+ \):

\[
\begin{align*}
\text{Base} & \quad \varphi_{t+s}(x) \leftarrow \varphi_t(x), Q_s(x). \\
\text{Recursion} & \quad \varphi_{t+s}(x) \leftarrow \varphi_{t+s}(x).
\end{align*}
\]

The next example shows the transformation of the regular expression \((ab^+)+c\) to M-datalog.

**Example 36 (Translation of regular expression to Monadic Datalog)**

Translation of the regular expression \((ab^+)+c\) to M-datalog

The next program is the M-datalog for the expression \( ab^+ \).

\[
\begin{align*}
\varphi_b(x) & \leftarrow P_b(x). \\
\varphi_{b^+}(x) & \leftarrow \varphi_b(x). \\
Q_{b^+}(x) & \leftarrow S(x, y), \varphi_{b^+}(x). \\
\varphi_{b^+}(x) & \leftarrow P_b(x), Q_{b^+}(x). \\
\varphi_{ab^+}(x) & \leftarrow P_a(x), Q_{b^+}(x).
\end{align*}
\]

Assume \( r \) be any regular expression and \( \varphi_r \), the corresponding IDB predicate, the transformation of the concatenation of \( ab^+ \) and \( r \) is as follows:
\[ Q_r(x) \triangleq S(x, y), \varphi_r(y). \]
\[ \varphi_{br}(x) \triangleq P_b(x), Q_r(x). \]
\[ Q_{b+r}(x) \triangleq S(x, y), \varphi_{b+r}(x). \]
\[ \varphi_{b+r}(x) \triangleq P_b(x), Q_{b+r}(x). \]
\[ \varphi_{ab+r}(x) \triangleq P_a(x), Q_{b+r}(x). \]

The next program is the M-datalog for the expression \((ab+)\). 

\[ \varphi_{(ab+)}(x) \triangleq \varphi_{ab+}(x). \]
\[ \varphi_{(ab+)+}(x) \triangleq \varphi_{ab+((ab+)\+)}(x). \]

Note that the program of the predicate \(\varphi_{ab+((ab+)\+)}(x)\) can be expanded by using program B.2 by replacing \(r\) with \(((ab+)\+).\)

Finally we obtain the program for \((ab+)\)+c:

\[ \varphi_c(x) \triangleq P_c(x). \]
\[ Q_c(x) \triangleq S(x, y), \varphi_c(y). \]
\[ \varphi_{(ab+)c}(x) \triangleq \varphi_{ab+}(x), Q_c(x). \]
\[ \varphi_{(ab+)c}(x) \triangleq \varphi_{ab+((ab+)\+c)}(x). \]

where the IDB \(\varphi_{ab+((ab+)\+c)}\) is given as follows:

\[ Q_{((ab+)\+c)}(x) \triangleq S(x, y), \varphi_{((ab+)\+c)}(y). \]
\[ \varphi_{b((ab+)\+c)}(x) \triangleq P_b(x), Q_{((ab+)\+c)}(x). \]
\[ Q_{b+((ab+)\+c)}(x) \triangleq S(x, y), \varphi_{b+((ab+)\+c)}(x). \]
\[ \varphi_{b+((ab+)\+c)}(x) \triangleq P_b(x), Q_{b+((ab+)\+c)}(x). \]
\[ \varphi_{ab+((ab+)\+c)}(x) \triangleq P_a(x), Q_{b+((ab+)\+c)}(x). \]

As far as the other direction is concerned, there is a folklore theorem stating that MSO and M-datalog are equivalent in their power to define unary queries.

**Proposition 13 (Folklore [GK02a])** Over arbitrary finite structures, each M-datalog query is definable with MSO.
CLISTS

List of Theorems.

<table>
<thead>
<tr>
<th>Theorem</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theorem 1</td>
<td>Conjunctive Query Containment</td>
<td>9</td>
</tr>
<tr>
<td>Lemma 1</td>
<td>Containment of CQs [CM77]</td>
<td>15</td>
</tr>
<tr>
<td>Theorem 2</td>
<td>Containment of unions of CQs [SY80]</td>
<td>16</td>
</tr>
<tr>
<td>Theorem 3</td>
<td>Containment of CQs with Inequalities [ZÖ97]</td>
<td>17</td>
</tr>
<tr>
<td>Proposition 1</td>
<td>Acyclicity and join tree [AHV95]</td>
<td>19</td>
</tr>
<tr>
<td>Proposition 2</td>
<td>Satisfiability of CQ⁻</td>
<td>26</td>
</tr>
<tr>
<td>Lemma 2</td>
<td>Containment of CQs [CM77]</td>
<td>26</td>
</tr>
<tr>
<td>Lemma 3</td>
<td>SP of CQ⁻'s</td>
<td>27</td>
</tr>
<tr>
<td>Proposition 3</td>
<td>Necessary condition for containment of CQ⁻'s</td>
<td>27</td>
</tr>
<tr>
<td>Theorem 4</td>
<td>Necessary condition 2 for containment of CQ⁻'s</td>
<td>27</td>
</tr>
<tr>
<td>Theorem 5</td>
<td>Containment of CQ⁻'s</td>
<td>28</td>
</tr>
<tr>
<td>Lemma 4</td>
<td>Unions of CQ⁻'s</td>
<td>33</td>
</tr>
<tr>
<td>Lemma 5</td>
<td>Containment of unions of CQ⁻'s</td>
<td>33</td>
</tr>
<tr>
<td>Theorem 6</td>
<td>Containment of unions of CQ⁻'s</td>
<td>34</td>
</tr>
<tr>
<td>Theorem 7</td>
<td></td>
<td>34</td>
</tr>
<tr>
<td>Proposition 4</td>
<td>Complexity of CQ⁻ containment [Ull97]</td>
<td>41</td>
</tr>
<tr>
<td>Proposition 5</td>
<td>Containment of acyclic CQ⁻ containment, upper bound</td>
<td>41</td>
</tr>
<tr>
<td>Theorem 8</td>
<td>Containment of acyclic CQ⁻ containment, lower bound</td>
<td>42</td>
</tr>
<tr>
<td>Theorem 9</td>
<td>Reduction of disjunctive expansion to minimal model semantics</td>
<td>49</td>
</tr>
<tr>
<td>Proposition 6</td>
<td>Termination of disjunctive expansion</td>
<td>49</td>
</tr>
<tr>
<td>Corollary 1</td>
<td>Containment of CQs under DRC</td>
<td>50</td>
</tr>
<tr>
<td>Theorem 10</td>
<td>Containment of CQs under DRC</td>
<td>50</td>
</tr>
<tr>
<td>Theorem 11</td>
<td>Containment of CQs under DRC</td>
<td>50</td>
</tr>
<tr>
<td>Theorem 12</td>
<td>Equivalence of star-free regular language and FO</td>
<td>70</td>
</tr>
<tr>
<td>Theorem 13</td>
<td>Equivalence of PLTL and FO</td>
<td>71</td>
</tr>
<tr>
<td>Proposition/Statement</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>Proposition 7 Transformation of XP(//,*//) to star-free regular expression</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>Theorem 14 Normal Form of XP(//,*//)</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>Lemma 6</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>Lemma 7</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>Theorem 15 Containment of XP(//,*//) expressions</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>Theorem 16</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>Theorem 17 Complexity</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>Lemma 8</td>
<td>89</td>
<td></td>
</tr>
<tr>
<td>Theorem 18 Query evaluation of acyclic CQ</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>Lemma 9 Query evaluation of $\mathcal{CQLC}^f$</td>
<td>90</td>
<td></td>
</tr>
<tr>
<td>Lemma 10 Containment of $\mathcal{CQLC}^f$ queries</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>Theorem 19</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>Theorem 20</td>
<td>93</td>
<td></td>
</tr>
<tr>
<td>Theorem 21</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>Theorem 22 Homomorphism of $\mathcal{CQLC}^f$'s with forward axes</td>
<td>98</td>
<td></td>
</tr>
<tr>
<td>Lemma 11</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>Theorem 23 Containment of XPath fragment with forward axes</td>
<td>115</td>
<td></td>
</tr>
<tr>
<td>Proposition 8 A necessary condition</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>Lemma 12</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>Lemma 13</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>Theorem 24</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>Theorem 25 Containment of $\mathcal{CQLC}^f$ queries with filter</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>Theorem 26 Complexity of satisfiability for PLTL [SC85]</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>Theorem 27 Complexity of model checking for PLTL [SC85]</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>Theorem 28 Complexity of model checking for CTL [Eme90]</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>Theorem 29 Complexity of satisfiability for CTL [Eme90]</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>Theorem 30 Complexity of model checking for CTL* [Eme90]</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>Theorem 31 Complexity of satisfiability for CTL* [Eme90]</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>Proposition 9 Decidability of SnS [Rab69]</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>Theorem 32 Bisimulation invariance of modal logic [vB83]</td>
<td>137</td>
<td></td>
</tr>
<tr>
<td>Proposition 10 Bisimulation invariance [HM85]</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>Proposition 11 Tree model property</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>Theorem 33 Equivalence of MSO and finite automata on words</td>
<td>141</td>
<td></td>
</tr>
</tbody>
</table>
List of Definitions

<table>
<thead>
<tr>
<th>Theorem/Proposition</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theorem 34</td>
<td>Equivalence of Monadic Datalog and finite automata</td>
<td>141</td>
</tr>
<tr>
<td>Proposition 12</td>
<td>Equivalence of Monadic Datalog and finite automata</td>
<td>142</td>
</tr>
<tr>
<td>Proposition 13</td>
<td>Folklore [GK02a]</td>
<td>145</td>
</tr>
</tbody>
</table>

List of Definitions.

<table>
<thead>
<tr>
<th>Definition</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition 1</td>
<td>Query equivalence</td>
<td>3</td>
</tr>
<tr>
<td>Definition 2</td>
<td>Query containment</td>
<td>3</td>
</tr>
<tr>
<td>Definition 3</td>
<td>Containment and equivalence</td>
<td>3</td>
</tr>
<tr>
<td>Definition 4</td>
<td>Query containment and equivalence</td>
<td>15</td>
</tr>
<tr>
<td>Definition 5</td>
<td>Union of CQs [Ull89]</td>
<td>16</td>
</tr>
<tr>
<td>Definition 6</td>
<td>Super-Positive SP $Q^+$</td>
<td>27</td>
</tr>
<tr>
<td>Definition 7</td>
<td>Unions of $CQ$'s</td>
<td>33</td>
</tr>
<tr>
<td>Definition 8</td>
<td>Acyclic $CQ$'s</td>
<td>40</td>
</tr>
<tr>
<td>Definition 9</td>
<td>Implication Constraint</td>
<td>46</td>
</tr>
<tr>
<td>Definition 10</td>
<td>Disjunctive Referential Constraint</td>
<td>47</td>
</tr>
<tr>
<td>Definition 11</td>
<td>Consistency</td>
<td>47</td>
</tr>
<tr>
<td>Definition 12</td>
<td>Containment under $DRC$ and $IC$</td>
<td>47</td>
</tr>
<tr>
<td>Definition 13</td>
<td>Disjunctive Referential Expansion</td>
<td>48</td>
</tr>
<tr>
<td>Definition 14</td>
<td>Acyclicity of referential constraints</td>
<td>49</td>
</tr>
<tr>
<td>Definition 15</td>
<td>Document Order</td>
<td>57</td>
</tr>
<tr>
<td>Definition 16</td>
<td>XML Axes</td>
<td>60</td>
</tr>
<tr>
<td>Definition 17</td>
<td>Regular expression</td>
<td>67</td>
</tr>
<tr>
<td>Definition 18</td>
<td>Regular language</td>
<td>68</td>
</tr>
<tr>
<td>Definition 19</td>
<td>Star-free regular expression</td>
<td>68</td>
</tr>
<tr>
<td>Definition 20</td>
<td>$FO[&lt;]$</td>
<td>70</td>
</tr>
<tr>
<td>Definition 21</td>
<td>Normal Form of XP(/,//,*) Expressions</td>
<td>73</td>
</tr>
<tr>
<td>Definition 22</td>
<td>Containment mapping of regular expressions</td>
<td>74</td>
</tr>
<tr>
<td>Definition 23</td>
<td></td>
<td>89</td>
</tr>
<tr>
<td>Definition 24</td>
<td>Invalid Mapping</td>
<td>93</td>
</tr>
<tr>
<td>Definition 25</td>
<td>Homomorphism</td>
<td>97</td>
</tr>
<tr>
<td>Definition 26</td>
<td></td>
<td>109</td>
</tr>
<tr>
<td>Definition 27</td>
<td>Tree Database</td>
<td>116</td>
</tr>
<tr>
<td>Definition 28</td>
<td>Tree Containment of $CQLC^t$ queries</td>
<td>116</td>
</tr>
<tr>
<td>Definition 29</td>
<td>Wildcards Elimination</td>
<td>118</td>
</tr>
</tbody>
</table>
List of Examples.

Example 1  Containment mapping ........................ 16
Example 2  Containment mapping ........................ 17
Example 3  Acyclic CQ  .................................... 18
Example 4  Conjunctive query with the chain form .... 19
Example 5  Datalog with transitive closure .......... 20
Example 6  Monadic Datalog ............................ 20
Example 7  Containment of $CQ^\sim$'s ................. 23
Example 8  Containment of $CQ^\sim$'s ................. 24
Example 9  Containment of CQ's ...................... 26
Example 10  Containment of $CQ^\sim$'s ............. 29
Example 11  Containment of $CQ^\sim$'s ............. 30
Example 12  Containment of $CQ^\sim$'s ............. 32
Example 13  Containment of $CQ^\sim$'s ............. 32
Example 14  Containment of unions of $CQ^\sim$'s ...... 33
Example 15  Containment of unions of $CQ^\sim$'s ...... 34
Example 16  Query rewriting with unions of $CQ^\sim$'s .. 35
Example 17  Query rewriting with unions of $CQ^\sim$'s .. 35
Example 18  Query rewriting with unions of $CQ^\sim$'s .. 36
Example 19  Acyclic $CQ^\sim$ ........................... 40
Example 20  A relational database schema .......... 45
Example 21  Containment of CQ's under disjunctive integrity constraints 48
Example 22  Containment of CQ's under $DRC$ .......... 50
Example 23  Star-free languages to first-order sentences 70
Example 24  ................................................ 77
Example 25  Containment checking ..................... 80
Example 26  ................................................ 88
Example 27  ................................................ 92
Example 28  ................................................ 95
Example 29  ................................................ 98
Example 30  ............................................... 110
List of Algorithms

<table>
<thead>
<tr>
<th>Example</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 31</td>
<td></td>
<td>113</td>
</tr>
<tr>
<td>Example 32</td>
<td></td>
<td>115</td>
</tr>
<tr>
<td>Example 33</td>
<td>Tree containment</td>
<td>117</td>
</tr>
<tr>
<td>Example 34</td>
<td>Wildcard elimination</td>
<td>118</td>
</tr>
<tr>
<td>Example 35</td>
<td>Guarded Logic and Acyclic CQ</td>
<td>139</td>
</tr>
<tr>
<td>Example 36</td>
<td>Translation of regular expression to Monadic Datalog</td>
<td>144</td>
</tr>
</tbody>
</table>

List of Figures.

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1</td>
<td>The flight example</td>
<td>6</td>
</tr>
<tr>
<td>Figure 1.2</td>
<td>A family tree</td>
<td>7</td>
</tr>
<tr>
<td>Figure 1.3</td>
<td>Queries (a), (b) and (c) are tree equivalent.</td>
<td>7</td>
</tr>
<tr>
<td>Figure 2.1</td>
<td>The hypergraph and join tree of Example 3</td>
<td>19</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>The graphic illustration of Example 11.</td>
<td>31</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>The graphic illustration of Example 13.</td>
<td>33</td>
</tr>
<tr>
<td>Figure 6.1</td>
<td>A XML tree</td>
<td>56</td>
</tr>
<tr>
<td>Figure 6.2</td>
<td>Inverted list encoding</td>
<td>58</td>
</tr>
<tr>
<td>Figure 6.3</td>
<td>Dewey Order encoding</td>
<td>59</td>
</tr>
<tr>
<td>Figure 6.4</td>
<td>An XML satisfies the DTD</td>
<td>64</td>
</tr>
<tr>
<td>Figure 7.1</td>
<td>XP(//,/*) queries and the corresponding star-free expressions</td>
<td>72</td>
</tr>
<tr>
<td>Figure 7.2</td>
<td>Containment Checking of Two XP(//,/*) Queries</td>
<td>82</td>
</tr>
<tr>
<td>Figure 8.1</td>
<td>A sample XML document fragment (a) and the corresponding tree representation (b).</td>
<td>86</td>
</tr>
<tr>
<td>Figure 8.2</td>
<td>A 5-width query decomposition of query the in Example 26</td>
<td>90</td>
</tr>
<tr>
<td>Figure 8.3</td>
<td>An invalid mapping</td>
<td>99</td>
</tr>
<tr>
<td>Figure 8.4</td>
<td>An invalid mapping</td>
<td>99</td>
</tr>
<tr>
<td>Figure 8.5</td>
<td>An invalid mapping</td>
<td>99</td>
</tr>
<tr>
<td>Figure 8.6</td>
<td>An invalid mapping</td>
<td>99</td>
</tr>
<tr>
<td>Figure 8.7</td>
<td>Mapping from a descendant axis to a following axis is an invalid mapping. The constraints with underlined variables cause a contradiction.</td>
<td>101</td>
</tr>
<tr>
<td>Figure 8.8</td>
<td>Mapping from a child axis to a following axis is an invalid mapping. The constraints with underlined variables cause a contradiction.</td>
<td>102</td>
</tr>
</tbody>
</table>
Figure 8.9  Mapping from a following axis to a descendant axis is an invalid mapping. The constraints with underlined variables cause a contradiction. ................................................. 102
Figure 8.10 Mapping from a following axis to a child axis is an invalid mapping. The constraints with underlined variables cause a contradiction. ................................................. 102
Figure 8.11 Possible mappings from a vertical axis ................................. 105
Figure 8.12 The inequality graph concerning begin and end variables of the query a/child :a//desc :a//following :a//child :a. ............... 108
Figure 8.13 The resulting inequality graph after adding all the possible constraints with the form $E_u \geq B_v$, where $u < v$. ................. 108
Figure 8.14 The inequality graph concerning level variables of the query $a//desc :a//desc :a//following :a//desc :a$. ......................... 108
Figure 8.15 The resulting inequality graph after adding all the possible constraints with the form $L_u \geq B_v$, where $u < v$. ................. 108
Figure 8.16 Queries (a), (b) and (c) are tree equivalent. ......................... 117
Figure 8.17 Two representations of a sample Core XPath query, which can be distinguished if the ordered query is considered. ............... 123
Figure 9.1 The relationship between fragments of CQs and their containment checking .......................................................... 129
Figure A.1 The Kripke Structure $M$. ............................................. 131
Figure A.2 Two bisimilar Kripke Structures ...................................... 137
Figure A.3 Two Kripke Structures Which are not Bisimilar .................... 137
Figure A.4 The Kripke Structure $M'$ ............................................. 138
Figure A.5 The Join tree .......................................................... 140

List of Tables.
Table 3.1 The five canonical databases and their answers to $Q_1$ and $Q_2$ . . 24
Table 4.1 The Overview ..................................................... 41
Table 8.1 The mapped subgoals in $Q_1$ of Example 31 .......................... 115
Table 8.2 The mapped subgoals in $Q_1$ of Example 32 .......................... 116
BIBLIOGRAPHY

Bibliography


the EATCS, 68:90–103, 1999. (pp.138, 139)

[GS03] M. Grohe and N. Schweikardt. Comparing the succinctness of monadic
query languages over finite trees. In Proceedings of the 17th Interna-
tional Workshop on Computer Science Logic (CSL’03), 2003. (p.141)

[GS04] M. Grohe and N. Schweikardt. The succinctness of first-order logic on
linear orders. In LICS, 2004. (p.68)

[GSUW94] A. Gupta, Y. Sagiv, J. D. Ullman, and J. Widom. Constraint Checking
with Partial Information. In Proceedings of the Thirteenth Symposium

[GSW96] S. Guo, W. Sun, and M. A. Weiss. On Satisfiability, Equivalence,
and Implication Problems Involving Conjunctive Queries in Database

RWTH Aachen, 2002. (p.138)

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and

Queries Under Functional and Inclusion Dependencies. In Proceed-
ings of the ACM Symposium on Principles of Database Systems. ACM
Press, 1982. (pp.5, 21, 45)

[Kam68] J. A. W. Kamp. Tense Logic and the Theory of Linear Order. PhD
thesis, University of California, Los Angeles, 1968. (p.71)

[KLP00] G. Kuper, L. Libkin, and J. Paredaens. Constraint Databases. Springer,
2000. (p.85)


Reachability, and Satisfiability in Datalog Extensions. In Proceed-
ings of the Twelfth ACM PODS. ACM Press, 1993. (p.90)

2000. (p.35)

Programming. MIT press, Cambridge, Massachusetts, 1992. (pp.46,
47, 127)

[LMS95] A. Y. Levy, A. O. Mendelzon, and Y. Sagiv. Answering queries us-


[NS03] F. Neven and T. Schwentick. XPath Containment in the Presence of Disjunction, DTDs, and Variables. In ICDT, 2003. (pp 10, 85)


[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear


[vB83] J. van Benthem. *Modal Logic and Classical Logic.* Bibliopolis, Napoli,
1983. (pp137, 148)


