ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής

Διαχείριση Πληροφορίας και Αβεβαιότητας σε Περιβάλλον Πολλαπλών Ετερογενών Πηγών Πληροφόρησης.

Διδακτορική Διατριβή

Άγγελος - Γεώργιος Βασιλακόπουλος

Επιβλέπουσα Καθηγήτρια:
Φώτω Αφράτη
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Οκτώβριος 2014

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής

Διαχείριση Πληροφορίας και Αβεβαιότητας σε Περιβάλλον Πολλαπλών Ετερογενών Πηγών Πληροφόρησης.

Διδακτορική Διατριβή

Άγγελος - Γεώργιος Βασιλακόπουλος

Τριμελής Συμβουλευτική Επιτροπή:

1. **Φώτω Αφράτη, Καθ. Ε.Μ.Π. (επιβλέπουσα)**
2. Ιωάννης Βασιλείου, Καθ. Ε.Μ.Π.
3. Μανόλης Γεργατσούλης, Αν. Καθ. Ιόνιο Παν.

Επταμελής Εξεταστική Επιτροπή:

1. **Φώτω Αφράτη, Καθ. Ε.Μ.Π. (επιβλέπουσα)**
2. Ιωάννης Βασιλείου, Καθ. Ε.Μ.Π.
3. Μανόλης Γεργατσούλης, Αν. Καθ. Ιόνιο Παν.
4. Νικόλαος Παπασπύρου, Αν. Καθ. Ε.Μ.Π.
5. Δημήτριος Φωτάκης, Επίκ. Καθ. Ε.Μ.Π.
6. Ιζαμπώ Καράλη, Επίκ. Καθ. Ε.Κ.Π.Α.
7. Παναγιώτης Σταματόπουλος, Επίκ. Καθ. Ε.Κ.Π.Α.

Αθήνα, Οκτώβριος 2014.

....

Άγγελος - Γεώργιος Βασιλακόπουλος
Διδάκτωρ της Σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Ε.Μ.Π.

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

Information and Uncertainty Management for Multiple Heterogeneous Data Sources

Ph.D. Thesis

Angelos - Georgios Vasilakopoulos

Supervisor:
Foto Afrati
Professor
NTUA

Athens, October 2014

Contents

List of Figures

Abstract

In this thesis we investigate five problems of databases with uncertainty and lineage. We start with the analysis for those databases of the *query containment* and *query equivalence* problems. We present and define five new kinds of semantics for *database containment* with uncertainty and lineage. We prove that the complexity of *query containment* for databases with uncertainty and lineage for *conjunctive queries (CQs)* and for *Unions of conjunctive queries (UCQs)* remains *NP-complete* as it is with ordinary databases for all five new kinds of containment semantics. We prove that the complexity of *query equivalence* for databases with uncertainty and lineage for CQs and UCQs is also *NP-complete* for the first two kinds of ULDB database containment and *Graph-Isomorphism-complete* for the last three. Finally we define five new "equality" semantics of ULDB containment and we show that they are important for ULDB data integration purposes. The complexity of checking conjunctive query containment under all these five kinds of "equality" ULDB containment is NP-complete.

Further we study and analyse the *data exchange* problem for databases with uncertainty and lineage. We present and define new logical semantics of *certain answers* for data with uncertainty and lineage. We present a new *u-chase* algorithm that extends the known *chase* algorithm which is about ordinary databases. We prove that the new *u-chase* algorithm can be used for *query answering* for conjunctive queries with the complexity of computing *certain answers* that remains low, i.e. polynomial time (Ptime), as with ordinary databases when the set of dependencies between the source and the target schema is a *weakly cyclic* set of *tuple generating dependencies- tgds.* We prove that when we also have target (equality generating dependencies - egds) then the problem of *query answering* for databases with uncertainty and lineage becomes *NP-hard* (in contrast with ordinary databases).

We next study and analyse the problem of query computing for conjunctive queries for databases with uncertainty and lineage when we attach belief values to uncertain data that come from a *possibility distribution.* We prove that the model of databases with uncertainty and lineage and with possibilis-

tic values is *closed* for conjunctive queries. This result solves the problem of previous results that show that models with uncertainty and possibilities (but no lineage) are not closed for conjunctive queries. We prove that in the model of databases with uncertainty and lineage and possibilities we can compute conjunctive queries (along with their possibility values) with low polynomial complexity (Ptime) in contrast with the high complexity #P of existing approaches where uncertainty values are probabilistic.

The fourth problem that we study is the problem of how to *efficiently* compute *aggregate queries* and specifically SUM queries, posed on *big data*. We use a small useful lineage instead of the initial big data. We present a new Algorithm *Comp-Lineage* which computes in polynomial time an *Aggregate Lineage* with small size that is independent of the size of the initial big data. We prove that this small *Aggregate Lineage* can be used to approximate well any SUM query whose value is large, with time complexity that depends only on the small size of the *Aggregate Lineage* and hence independent with the large size of the initial big data.

Finally we study, analyse and implement in the parallel environment *MapReduce* the computation of 2-Way Joins for *big data* that may also be *skewed*. We present a new algorithm, suitable for MapReduce, which computes 2-Way Joins and can efficiently handle *data skewness* in contrast with existing algorithms. We prove that our algorithm matches the lower communication cost bound. We further implement in Java/Hadoop our algorithm with experiments whose communication cost verifies the theoretical cost.

Chapter 1

Introduction

One of the main important requirements of current research is to integrate information distributed in different sources that are moreover heterogeneous [ALP08a, ALP08b, AK08, FKK10, GKIT07, SDH, MM09]. Data does not anymore come from a single source, for example from a company, but rather often from many different sources or over the internet. For this reason, research has focused on the relevant problems of *data integration* and *data exchange*. Research has also shown that in these problems uncertainty rises, due to the fact that the different sources are heterogeneous [FKMP05, ALP08a]. In addition many recent applications involve uncertain data [AKG91, BGMP92, DS04, IL84, SUW09]. Hence, we need to be able to efficiently manage data with *uncertainty*. In a setting where the answer of a query can come from the reconciliation of data existing in different sources, it is particularly important for us to be able to record from where the answer came from, in other words the *lineage* or *provenance* of the data [BSH$^+$08a]. In many other applications it is necessary to record the lineage of the data [BKT01, BT07, CW00, CW03, GKT07]. We name a few such applications: bio-informatics, analysis of scientific data or of data coming from the web. The need to record lineage is crucial especially today for the following additional reason: the computational power we have nowadays has given us the ability to analyse huge amounts of data, i.e., *big data*. It is important to be able to extract from such big data any useful information that may exist in them (for example by using techniques from *machine learning*) [ora13a, ora13b]. In such a context, a small set of useful data that can give us information about the answer of a query that is initially posed on big data becomes the lineage of the answer [AFV14].

In this new setting of many heterogeneous data sources, that moreover may include big or uncertain data, it is clear the need of efficiently managing databases with uncertainty and study the problem of data exchange. In addition it is important to be also able to record the lineage of the data. Both these two properties of uncertainty and lineage are only externally supported in the classic ordinary *relational data model*. On the other hand, a new data model which internally supports uncertainty and lineage has been recently presented (the *ULDB - Uncertainty and Lineage DataBase* data model) [BSH$^+$08a]. One of the main reasons for its creation is the that it would be useful in data exchange and data integration problems in which uncertainty arises. Uncertainty in the ULDB model is expressed through a set of possible alternative values (*alternatives*) that a tuple can have. In addition, the problem of *query computing* for *conjunctive queries*, that is the SELECT, PROJECT, JOIN subset of SQL, was studied for this model and showed that its data time complexity is polynomial (*Ptime*) [BSH$^+$08a]. Further, an extension of this model was studied in which each alternative value can be assigned with a probability which represents our belief that this alternative is the correct one. In this case the computation of the probabilities of the answers of conjunctive queries is a hard problem, with complexity #P [DS04, GT06, RPT11, STW08].

This thesis concerns the study of five general problems:

i) Firstly, we study the problems of *query containment* and *query equivalence* for this new model of data with uncertainty and lineage. The study of these problems is related and needed for the second problem of *data exchange* that we study. These two are important problems that have gained a lot of attention in database research [ALU07, AK10, ALM06, Ull97, FKMP05, CM77]. Most research however, focuses on ordinary databases with no uncertainty nor lineage recording. In the present thesis five new semantics of query containment are presented for databases with uncertainty and lineage and it is shown that the complexity of the query containment for conjunctive queries (CQs) and for unions of conjunctive queries (UCQs) retain the same NP-complete data complexity as in the case of ordinary databases with no uncertainty [CM77]. We move on to the problem of query equivalence and we show that for both CQs and UCQs the complexity for the first three kinds of ULDB containment remains NP-complete, while for the last two is Graph-Isomorphism-complete. On another perspective database containment was defined in [ASUW10] for uncertain databases without lineage. We also investigate CQ query containment under this definition, give corresponding new semantics and show that they are useful in ULDB data integration and that the complexity remains NP-complete.

ii) In contrast, when we move to the *query answering* problem for conjunctive queries in a *data exchange* setting with heterogeneous source and target schemas and where data (both source and target) have uncertainty and lineage, we show in which cases the complexity remains polynomial (Ptime) as with ordinary databases and when it increases to NP-hard. More specifically, we study and analyse the *data exchange* problem for databases with uncertainty and lineage. We present and define new logical semantics of *certain answers* for data with uncertainty and lineage. We present a new *u-chase* algorithm that extends the known *chase* algorithm which is about ordinary databases. We prove that the new *u-chase* algorithm can be used for *query answering* for conjunctive queries with the complexity of computing *certain answers* that remains low, i.e. polynomial time (Ptime), as with ordinary databases when the set of dependencies between the source and the target schema is a *weakly cyclic* set of *tuple generating dependencies- tgds*. We prove that when we also have target (equality generating dependencies - egds) then the problem of *query answering* for databases with uncertainty and lineage becomes *NP-hard* (in contrast with ordinary databases).

iii) We study the use of belief values other than probabilities that can be assigned in uncertain data so that the complexity of conjunctive query computing can remain polynomial (Ptime) in contrast with existing work that uses probabilities and has high complexity #P [DS04, GT06, RPT11, STW08].

iv) The fourth problem that we study concerns management of information on *big data*. This problem belongs to the very recent and important need of being able to do efficient computations and find crucial values for an answer posed on data with big size [ora13a, ora13b]. Specifically we study how the computation and recording of a useful lineage can be helpful to efficiently compute *aggregate queries* and specifically SUM queries on big data with low complexity using this small useful lineage instead of the initial big data. So, the solution that we propose is the management of this big-sized information to be done with the use of a new kind of suitable lineage. This lineage concerns the information of only a few important values which are a result of suitable computations over the many initial ones. We call this useful lineage as *Aggregate Lineage*. We present in this thesis a new algorithm that can compute Aggregate Lineage in polynomial time. We show that we can use Aggregate Lineage to approximate well the answer of any SUM query that can be posed on the initial big data. It is important to point out that our algorithm computes a small lineage which can be subsequently used to approximate any SUM query, which we do not need to know beforehand when we compute the Aggregate Lineage. In contrast, one Aggregate Lineage is suitable for every possible (and unknown) SUM query that has a large value.

v) Another approach to efficiently make computations on big data is the use of the parallel MapReduce environment. The fifth and last problem that we study concerns the design and implementation of an algorithm that can efficiently and with the least possible communication cost solve the computation of a 2-way Join in MapReduce even for cases where existing algorithms are inefficient due to data skewness.

1.1 A Motivating Example of Query Containment for Databases with Uncertainty and Lineage

The problem of *query containment* and *query equivalence* have applications in many important other problems of database research, like data integration and exchange and the use of views [ALU07, AK10, ALM06, Ull97, FKMP05]. Additionally it is very relevant with the *query optimisation* problem, which is one of the open problems for databases with uncertainty and lineage [BSH+08a]. A query Q_1 is contained in a query Q_2 if for every database D the answer of Q_1 is contained in the answer of Q_2 when they are posed on D. In ordinary databases the answer of a query is just a simple *set of tuples*, so database containment is just a test of simple set containment. In contrast, an uncertain database is not a set of tuples, rather it represents a set of databases that are its *possible instances - PIs*. Hence, even the simpler problem of database containment between two given uncertain database instances is not trivial for uncertain databases. In this research we study five different semantics for database containment with uncertainty and lineage, depending on how strict lineage information is required in the containment. We show in which cases which semantic is more suitable. For all semantics of database containment we have that data will be contained with a special kind of multiset (aka bag) containment: we can have many tuples with the same data if they have different lineage. The *query containment* problem in ordinary databases (that have no uncertainty nor lineage) has NP-complete data complexity for conjunctive queries [CM77]. Given that: i) The possible instances of a database with uncertainty and lineage are exponentially many and ii) conjunctive query containment for multisets is $\Pi_2^p - hard$ [ADG10, CV93], we would expect that conjunctive query containment belongs to a complexity class higher than NP-complete. In contrast, it is shown in this thesis that for all five different containment semantics the complexity of conjunctive query containment and equivalence for databases with uncertainty and lineage remains NP-complete. It is also shown that the same holds for Unions of Conjunctive Queries.

In more detail, we have that a database query Q defines a mapping from databases to databases. The problem of query containment is the following: A query Q_1 is said to be contained in a query Q_2 if for every database

D, database $Q_1(D)$ is contained in database $Q_2(D)$. So, query containment depends on the definition of database containment which, for ordinary databases, is defined as a simple *set containment* for each relation. This problem becomes more complicated for Databases with Uncertainty and Lineage (ULDBs): Each ULDB database relation is not anymore a simple set of tuples. Rather it represents a set of *possible instances* (PIs). Each possible instance is a Databases with only Lineage (LDB). In turn, each LDB is not a simple set of tuples but it also contains lineage information which is information from which other tuples this tuple is derived from. In an LDB relation we can have two tuples with the same data that point to different Lineage. As a result when investigating the problem of query containment for ULDBs we have to first define database containment for their possible instances, so for LDBs. Let us give a motiving example to illustrate why query containment becomes complicated for ULDBs:

Example 1. Consider the Database with Uncertainty and Lineage (ULDB) U of Figure 1.1. We will give technical details of the ULDB model in the next Chapter 2. For now let us describe what we need for our example: In the ULDB model each tuple with uncertainty, which is called an *x-tuple*, is a set of tuples called *alternatives*. The semantics of these alternatives is that at most one of them can be true, but we have uncertainty about which one. The ULDB model also captures lineage, i.e., from which alternative another one comes from. In order to succinctly represent lineage we attach an identifier on each x-tuple. For example suppose that we have the uncertain information that $John$ drives a $Honda$ or a $Mazda$ car, but we do not know which one. This information is stored in the two alternatives $(John, Honda)$ and $(John, Mazda)$ of the first x-tuple (which has identifier 11) of ULDB relation $Drives$. We refer to the first alternative $(John, Honda)$ of x-tuple 11 with the identifier $(11, 1)$. In our example the other two x-tuples 12 and 13 do not have uncertainty.

Suppose now that we have another ULDB relation Saw which includes the information that a witness saw a car near a crime-scene (this very suitable setting for explaining the ULDB model is also presented in [BSH+08b]). Let $Cathy$ be also uncertain about which car she saw, either a $Honda$ or a $Mazda$. This uncertainty is encoded in the two alternatives $(21, 1)$ and $(22, 2)$ of x-tuple 21 in ULDB relation Saw. If that was the case this ULDB database with the two relations Saw and $Drives$ would represent 4 possible instances, 2 for the two choices of x-tuple 11 and 2 for the two choices of x-tuple 21. But let us also suppose that alternative $(21, 1)$ comes from alternative $(11, 1)$ and

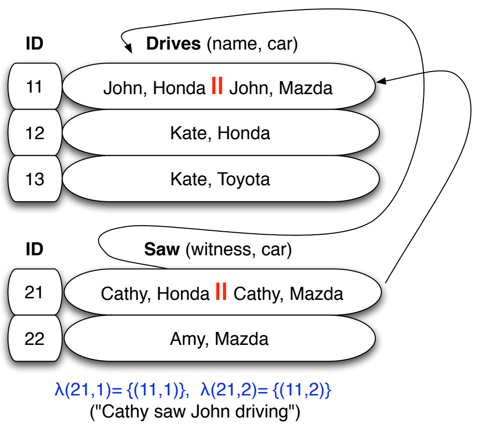

ID
Drives (name, car)
11
John, Honda || John, Mazda
12
Kate, Honda
13
Kate, Toyota
ID
Saw (witness, car)
21
Cathy, Honda || Cathy, Mazda
22
Amy, Mazda
λ(21,1)= {(11,1)}, λ(21,2)= {(11,2)}
("Cathy saw John driving")

Figure 1.1: The ULDB U of our motivating Example

that alternative $(21, 2)$ comes from alternative $(11, 2)$. In other words, that $Cathy$ saw a $Honda$ only if $John$ saw a $Honda$ or that she saw a $Mazda$ only if he drives a $Mazda$. This information is encoded with lineage λ, e.g. $\lambda(21, 1) = \{(11, 1)\}$ encodes the information that $(Cathy, Honda)$ can coexist in a Possible Instance only with $(John, Honda)$ (this information can mean for example that $Cathy$ additionally stated she saw $John$ driving). So now the ULDB has only two possible instances, for the two choices $(11, 1)$ and $(11, 2)$. In general the possible instances of a ULDB are indicated by the number of alternatives that do not point through lineage to other alternatives, i.e. that have empty lineage. Such alternatives are called *base*. We give more details about these logical restrictions that lineage poses in the possible instances of a ULDB in the next Chapter 2.

The two Possible Instances (let us denote them D_1 and D_2) of the ULDB of our example are shown in Figures 1.2 and 1.3 respectively. These possible instances do not have uncertainty but only lineage information, they are *database with lineage - LDBs*.

Consider now the following two Conjunctive Queries (CQs):

$Q_1(x) : -Drives(x, y), Saw(z, y)$ and
$Q_2(x) : -Drives(x, y)$

For ordinary databases, query Q_1 is contained in query Q_2 because there exists a containment mapping from Q_2 to Q_1 (which maps variables x and y to themselves).

But when these queries are posed on the ULDB of our Example this

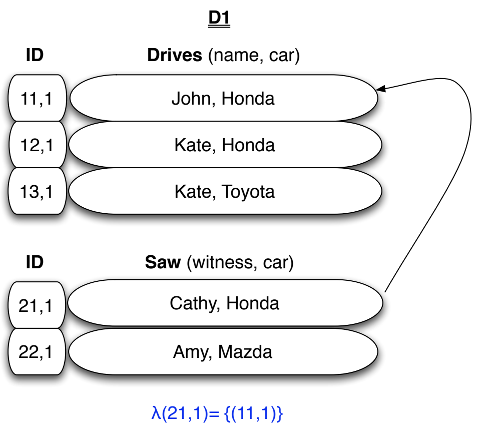


Figure 1.2: The first Possible Instance D_1 of our ULDB Example.

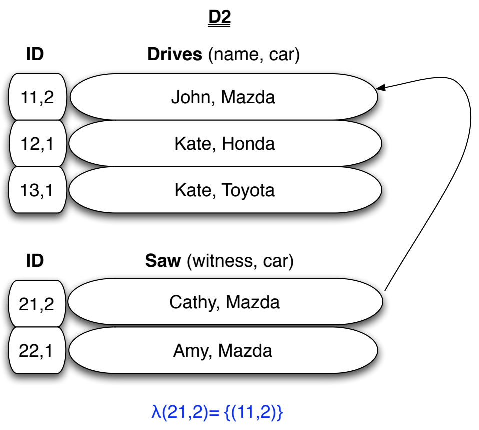


Figure 1.3: The second Possible Instance D_2 of our ULDB Example.

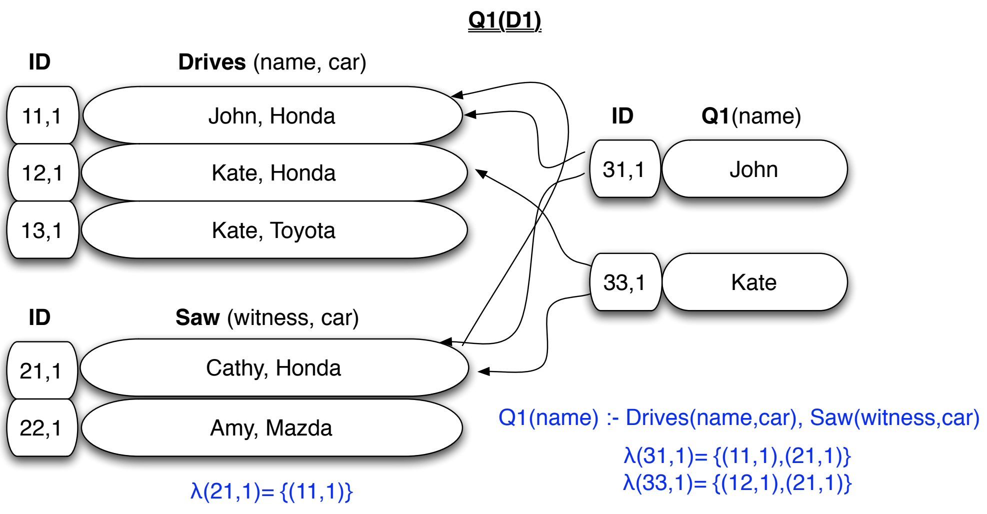


Figure 1.4: Query $Q_1(x) : -Drives(x, y), Saw(z, y)$ posed on the Possible Instance D_1 of Figure 1.2.

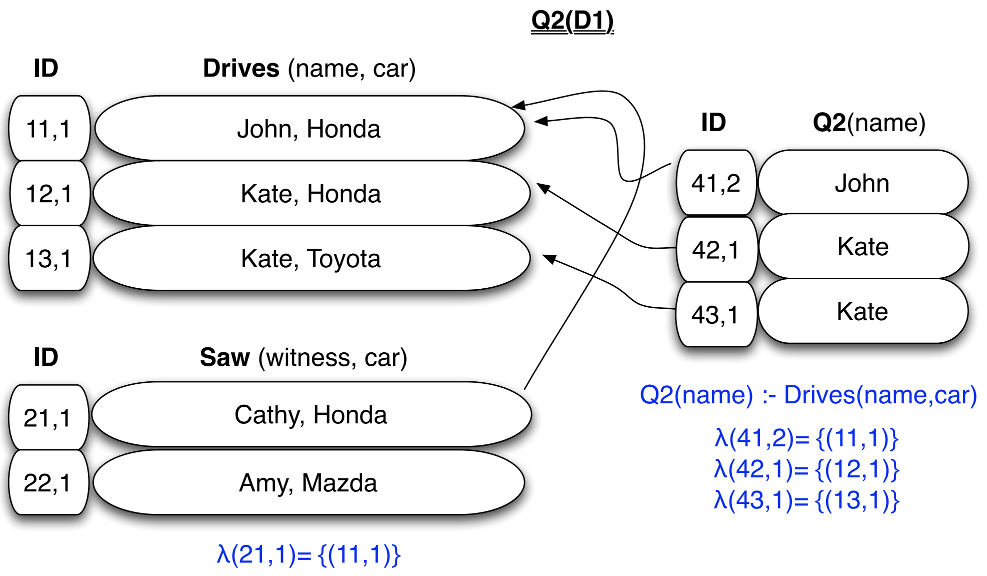


Figure 1.5: Query $Q_2(x) : -Drives(x, y)$ posed on the Possible Instance D_1 of Figure 1.2.

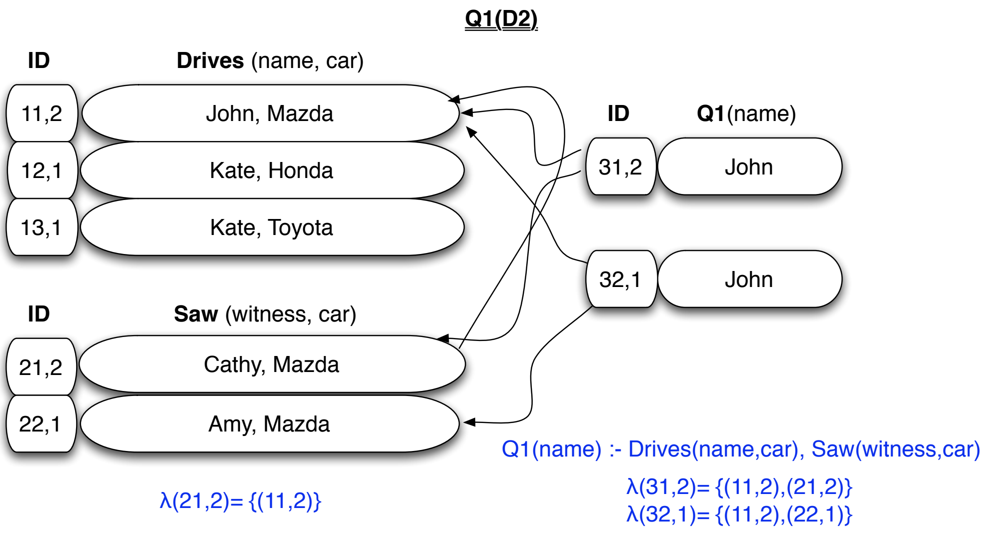


Figure 1.6: Query $Q_1(x) : -Drives(x, y), Saw(z, y)$ posed on the Possible Instance D_2 of Figure 1.3.

containment does not always hold. Consider the two possible instances of ULDB U. Since ULDBs and LDBs contain lineage information that points back to previous data, when we give the answer of a query we retain the relations over which the query is posed. The answer of Q_1 posed on D_1 is the LDB database shown in Figure 1.4 (along with relations `Drives` and `Saw` that have lineage pointing to them). The lineage information is natural: The tuple $(31, 1)$ in the answer comes from the information of tuples $(11, 1)$ and $(21, 1)$. That is, the answer *John* comes form the join of tuples $(John, Honda)$ and $(Cathy, Honda)$ and a projection on *John*.

Respectively Q_2 posed on D_1 is shown on Figure 1.5 and on D_2 on Figure 1.7. Intuitively Q_1 will return the name of a suspect (a driver that has been seen from a witness) while Q_2 will return the names of all drivers. Sometimes we care only about the fact that if someone is a suspect he must drive a car and hence say that Q_1 is contained in Q_2, like it holds for ordinary databases. Let us consider Q_1 and Q_2 posed on D_2. Indeed, if we care only about data, we see that the data *John* in relation Q_1 of the LDB database $Q_1(D_2)$ on Figure 1.6 is contained in the set of the data $\{John, Kate\}$ of Q_2 posed in the same instance D_2, shown on Figure 1.7. Alas, tuple *John* with identifier $31, 2$ on $Q_1(D_2)$ comes from both $11, 2$ and $21, 2$, while the tuple $41, 1$ with data *John* on $Q_2(D_2)$ comes only from $11, 2$. The semantics of LDB containment defined in the original work on ULDBs and LDBs [BSH+08a] define that two

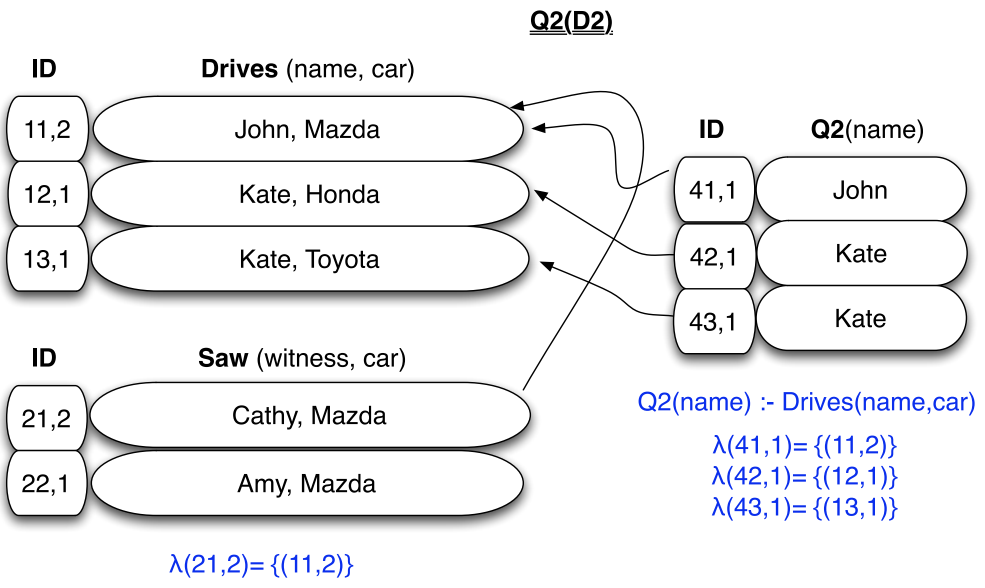


Figure 1.7: Query $Q_2(x) : -Drives(x, y)$ posed on the Possible Instance D_2 of Figure 1.3.

LDBs are contained if for each tuple in the contained relation we have a tuple with the same data in the containing relation and with lineage pointing to the same tuples through its transitive closure. It is easy to see that, under those semantics of LDB containment, Q_1 is NOT contained in Q_2 because e.g. $Q_1(D_2)$ is not contained in $Q_2(D_2)$ (nor in $Q_2(D_1)$). To see this we have that tuple $32, 1$ *John* of $Q_1(D_2)$ points to both $11, 2$ and $21, 2$, while the transitive closure of the tuple *John* on $Q_2(D_2)$ points only to $11, 2$.

On the other hand, as we pointed out, in some scenarios it will suffice that if someone is a suspect then he should also be a driver, ignoring the lineage information for LDB containment. Thus we present five different natural semantics for LDB database containment and explore in which practical cases each one is suitable. More specifically we define the following semantics of database containment for databases with uncertainty and lineage: i) Data, suitable if we only care about data, ii) CBase-Lineage, suitable in cases where we want to preserve data reliability, iii) TR-Lineage which preserves unreliability, iv) SBase-lineage which requires same base lineage, and the more strict v) Same-Lineage which requires exactly the same lineage. We give examples that illustrate why each different kind might be more suitable than the others. We study the exact interrelationship among them as concerns implication.

When we turn back to ULDB database containment, we define that a ULDB database is contained in another ULDB database if each of its Possible Instances of the first is LDB contained to a respective Possible Instance of the second. As a result, the five different semantics of LDB containment give us five different semantics of ULDB containment. Furthermore we define five new "equality" semantics of ULDB containment and we show that are important for ULDB data integration purposes.

We then can move on and revisit the notion of query containment for ULDBs by defining that a query Q_1 is ULDB contained in a query Q_2 if for every ULDB database U, database $Q_1(U)$ is ULDB contained in database $Q_2(U)$. Again we have five different semantics on ULDB query containment, following our LDB and ULDB database containment semantics.

We continue by investigating the complexity of the problem: We first find which containment test should hold for each kind of the five ULDB query containment semantics. Even though the definition of ULDB containment relies on the containment of its Possible Instances, we do not want a containment test which expensively first computes all the (exponentially many) Possible Instances of a ULDB. Rather we give a test on the queries themselves. As we saw, the containment test of ordinary databases which is a containment mapping from Q_2 to Q_1 not suffice for the LDB database containment semantics of [BSH+08a] and it does not suffice for all of our five semantics. The complexity of this containment mapping is NP-complete [CM77]. Interestingly, we prove that even with the most strict kind of semantics that require exactly the same lineage information in both contained and containing databases, the complexity of ULDB query containment for Connjunctive Queries (CQs) remains NP-complete. In detail, the first two kinds of containment have the ordinary containment mapping test, while the last three require a subgoal-onto containment mapping test. We continue and show that testing query containment for Unions of Conjunctive Queries (UCQs) is also NP-complete. Next we study the problem of query equivalence: For CQ and UCQ equivalence we prove that the complexity is NP-complete for the first two semantics and Graph-Isomorphism-complete for the last three. In [ASUW10] another kind of containment for uncertain databases with no lineage was discussed that was suitable for uncertain data integration purposes. We define five new corresponding "equality" semantics of ULDB containment and we show that are important for ULDB data integration purposes. The complexity of checking conjunctive query containment under all these five kinds of equality ULDB containment is NP-complete.

1.2 A Motivating Example of Data Exchange with Uncertainty and Lineage

After we investigated the query containment problem for Databases with Uncertainty and Lineage (ULDBs), we can move on and investigate the problem of query answering in a data exchange setting, which relies on query containment. One of the reasons for which the ULDB data model with uncertainty and lineage was created, was its application in the *data exchange* problem [BSH+08a]. This problem has been studied in the research for ordinary databases with no uncertainty nor lineage. It concerns the problem of transforming data that are expressed in a source schema so that they can be expressed through a target schema. The relation between source and target schemas is expressed with a set of dependencies. Even with ordinary databases with no initial uncertainty, the fact that source and target schemas are heterogeneous creates uncertainty during the data exchange [FKMP05, ALP08a]. It is hence expected to have higher complexity and additional uncertainty when even the initial data are uncertain (and moreover our model records lineage). One aspect of the data exchange problem concerns finding methods that can compute a target instance with polynomial complexity that is useful on query answering for queries expressed on the target schema. A second aspect concerns the definition and the semantics of these answers, so that they contain useful information (such answers are called *certain answers*). For ordinary databases it has been shown that computing certain answers for conjunctive queries has polynomial complexity if the dependencies between source and target schemas consist of a *weakly acyclic* set of *tuple generating dependencies (tgds)* and of *equality generating dependencies (egds)* [FKMP05]. Certain answer computation is done through an algorithm called *chase* which creates a target instance (called *universal*) on which queries can be posed. Chase algorithm concerns the dependencies between source and target schemas. Moving to a data exchange problem in which source data have uncertainty and lineage we now have two kinds of uncertainty: i) about which from the possible instances that the source data represents is the true one and ii) uncertainty due to the heterogeneous schemas. In this thesis: i) we introduce logical semantics for certain answers for data with uncertainty and lineage, ii) we present a new *u-chase* algorithm which extends the existing chase algorithms that was about ordinary databases, iii) we use the u-chase algorithm that we presented and we show that the complexity of computing certain answers for initial data with uncertainty and lineage remains low, specifically polynomial (Ptime) like with ordinary databases, when the set of source-to-target dependencies is a weakly acyclic set of tgds, while iv) when we also have a set of target egds then the problem becomes NP-hard (unlike the ordinary case).

In more detail: *Data exchange* is the problem of translating data that is described in a source schema to a different target schema. The relation between source and target schemas is typically defined by schema mappings. For ordinary databases with no uncertainty or lineage, it was shown in [FKMP05] that computing certain answers for conjunctive queries can be done with polynomial data complexity if the constraints satisfy specific conditions (are *weakly acyclic*). Recently the data exchange problem has been widely investigated, even for various uncertain frameworks, e.g., probabilistic [FKK10]. One aspect of the problem is finding procedures that compute in polynomial-time target instances that represent adequate (usually for query answering purposes) information. A challenging problem that has received considerable attention is the problem of giving meaningful semantics and computing answers of queries posed on the target schema of a data exchange setting [ALP08a, ALP08b, FKK10].

We present a data exchange framework that is capable of exchanging uncertain data with lineage and give meaningful certain answers on queries posed on the target schema. The data are stored in a *database with uncertainty and lineage (ULDB)* which represents a set of possible instances that are *databases with lineage (LDBs)*. Hence we need first to revisit all the notions related to data exchange for the case of LDBs. Producing all possible instances of a ULDB, like the semantics of certain answers would indicate, is exponential. We present a more efficient approach: a *u-chase* algorithm that extends the known chase procedure of traditional data exchange and show that it can be used to correctly compute certain answers for conjunctive queries in PTIME for a set of *weakly acyclic tuple generating dependencies (tgds)*. We further show that if we allow *equality generating dependencies (egds)* in the set of target constraints then computing certain answers for conjunctive queries becomes NP-hard.

More formally we have:
A data exchange problem $(\mathbf{S}, \mathbf{T}, \mathbf{\Sigma_{st}}, \mathbf{\Sigma_t})$ consists of a *source schema* $\mathbf{S}$, a *target schema* $\mathbf{T}$, a set Σ_{st} of source-to-target *dependencies*, and a set Σ_t of target dependencies.
The data exchange problem is the following:
Given a finite source instance I, find a finite target instance J such that $< I, J >= I \cup J$ satisfies Σ_{st} and J satisfies Σ_t. Such a J is called a *solution* for I.

We can now start from an Data Exchange example for ordinary databases and build up a similar database with lineage (LDB) and another similar

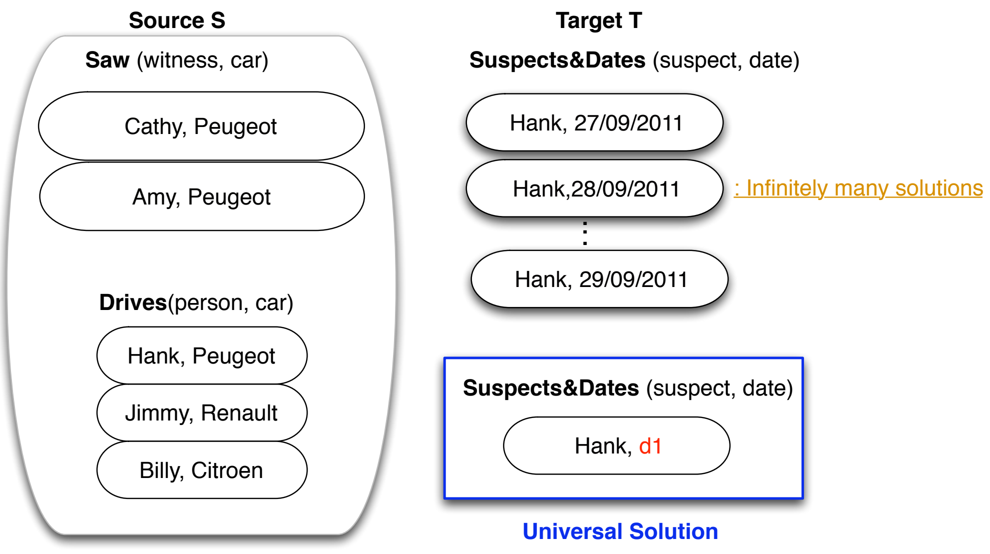


Figure 1.8: Data Exchange example with ordinary databases and the tgd dependency ξ : $Saw(witness, car)$, $Drives(p, car)$ $\rightarrow$ $\exists D$ $Suspects\&Dates(p, D)$.

database with uncertainty and lineage (ULDB) Data Exchange Example, to illustrate the new challenges:

Example 2. Suppose we have the ordinary source database S shown on Figure 1.8 and the set of dependencies consists of the single tgd:
$\xi : \texttt{Saw}(\texttt{witness}, \texttt{car}), \texttt{Drives}(\texttt{p}, \texttt{car}) \rightarrow \exists D\ \texttt{Suspects\&Dates}(\texttt{p}, \texttt{D})$.
In this Data Exchange setting, there exist infinite possible solutions. There exist a finite solution which represents all infinite possible solutions. Such a solution is called a *universal solution*. One universal solution $Suspects\&Dates$ is shown again in Figure 1.8. The value d_1 is a null value that can take any value. The tuple $(Hank, d_1)$ represents a set of infinitely many solutions, some of them are also depicted on Figure 1.8. The reason that we have uncertainty even in this case that we started with ordinary data arises from the heterogeneous schemas between the source and the target instances. This kind of uncertainty is represented with the null value d_1.

Consider now the very similar LDB data exchange scenario shown on Figure 1.9. When moving to LDBs, the ordinary universal solution $Suspects\&$ $Dates$ with only one tuple $(Hank, d_1)$, even though it represents an infinite number of tuples, is not suitable. The reason is that a tuple $(Hank, d_1)$ can be generated from the tgd dependency with two ways: i) From the com-

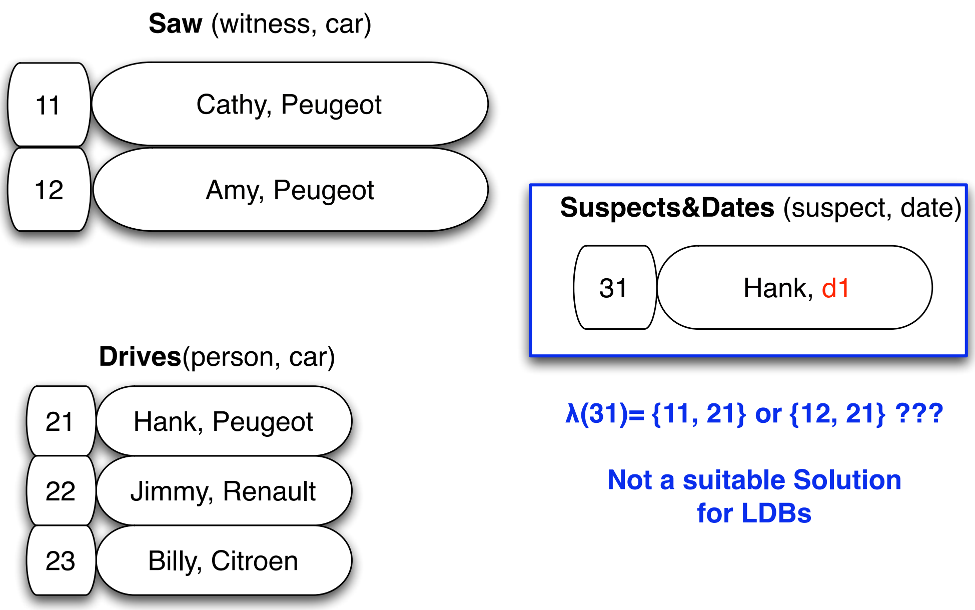


Figure 1.9: LDB Data Exchange example with the tgd dependency ξ : $Saw(witness, car)$, $Drives(p, car) \rightarrow \exists D \ \ Suspects\&Dates(p, D)$.

bination of tuples 11 and 21, i.e., $(Cathy, Peugeot)$ and $(Hank, Peuogeot)$ and ii) From the combination of tuples 12 and 21,i.e., $(Amy, Peugeot)$ and $(Hank, Peuogeot)$. Remember that LDBs capture where data comes from through lineage. If we use the ordinary universal solution with only one tuple $(Hank, d_1)$ we would not know which lineage to make it to point to.

The problem becomes even more complicated when having a ULDB source instance with both uncertainty and lineage. As we have said, a ULDB instance represents a set of LDBs which are its Possible Instances (PIs). Uncertainty is represented with alternative values. As a result a ULDB universal solution will now have two kinds of uncertainty:

- uncertainty about the *possible instances* that this source has (represented with alternatives and with lineage's logical restrictions)
- uncertainty that arises due to *heterogeneous* source and target schemas (represented with null values).

Hence, we need first to revisit the notion and give *suitable semantics* of tuple generating dependency - *tgd satisfaction* and of a *universal solution* for LDBs and ULDBs. Consider the ULDB Data Exchange example with the tgd

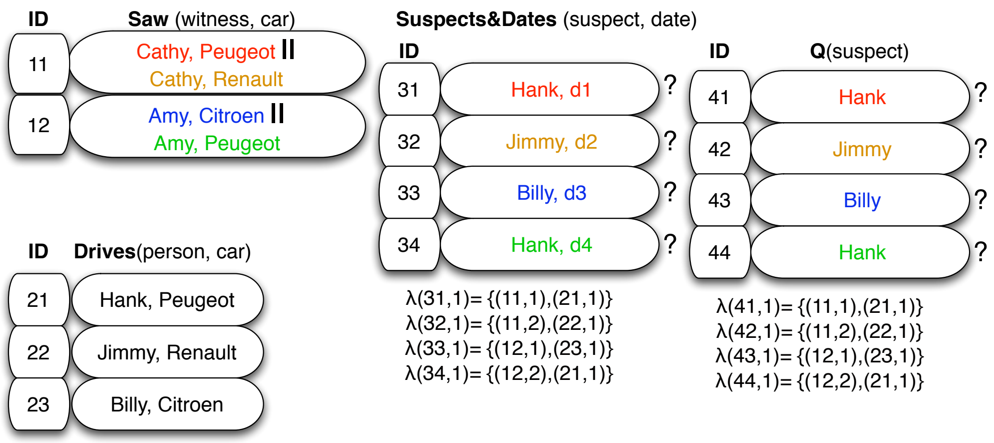


Figure 1.10: ULDB Data Exchange example with the tgd dependency ξ : $Saw(witness, car)$, $Drives(p, car)$ $\rightarrow$ $\exists D$ $Suspects\&Dates(p, D)$ and $Q(suspect)$:- $Suspects\&Dates(suspect, date)$

dependency $\xi : Saw(witness, car)$, $Drives(p, car)$ $\rightarrow$ $\exists D$ $Suspects\&Dates$ (p, D) on Figure 1.10. The source ULDB instance has 4 Possible Instances, 2×2 for the two alternative choices of x-tuples 11 and 12. We want to make sure that that a universal solution has the following property: for each Possible Instance of the source target, there exists a corresponding Possible Instance of the universal solution such that together they LDB satisfy the dependency ξ. Hence we need to first to investigate and define the LDB data exchange problem.

It is important to note that with ULDBs we cannot totally ignore lineage: Since a tgd generated tuples we have to make sure that a generated tuple will coexist in the correct/same Possible Instance. Thus, the data exchange notions will correspond to the LDB database containment semantic that requires contained base lineage. We further show that we can extend the known *chase* algorithm which was used for ordinary data exchange and present a new $u - chase$ algorithm that can compute ULDB universal solutions. Our u-chase algorithm has the correct semantics without having to exponentially compute all the Possible Instances of a ULDB. We define meaningful certain answers for queries posed on the target ULDB solutions. We show that the output of u-chase can be used to compute ULDB certain answers. The expected UDLDB universal solution and the ULDB certain answers for query $Q(suspect)$:- $Suspects\&Dates(suspect, date)$ are shown on Figure 1.10.

Finally, rather surprisingly, we prove that u-chase can be used to compute meaningful UDLB certain answers for conjunctive queries in polynomial time (Ptime), when the set of dependencies is a set of weakly acyclic tgds. So the computational complexity in this case remains the same as with ordinary databases. In contrast we prove that if the dependencies also contain equality generating dependencies (egds), the problem for ULDBs becomes NP-hard.

1.3 Efficient Query Computing when Combining Uncertainty, Lineage and Possibilities

For the problem of attaching belief values on uncertain data, if these values are probabilistic, then computing answers of conjunctive queries is a problem with high complexity #P [DS04, GT06, RPT11, STW08]. In constant we propose attaching belief values that come from a *possibility distribution*. The qualitative nature of possibilities, in contrast with the quantitive nature of probabilities, make them suitable for the kind of uncertainty that is described in the ULDB model for data with uncertainty and lineage. One important open problem of possibilities is that models with uncertainty and possibilities (and no lineage) is not *closed* on conjunctive queries. In contrast, we show that when we have the model with uncertainty, possibilities and lineage (i.e., ULDB with attached possibilities in the alternative values) is closed on conjunctive queries. In addition we also show that computing possibility values of the answers of conjunctive queries has low polynomial complexity (Ptime), which makes them more preferable than probabilities also for complexity reasons.

1.4 Computing Efficient Lineage for Aggregate Queries on Big Data

The problem of being able to efficiently compute out of *big data* which part is important is one of the main challenges of modern systems [ora13a, ora13b]. Many organisations have the infrastructure to maintain big structured data and need to find methods to efficiently discover patterns and relationships and to derive intelligence. Another growing fast application where data analytics are used to explain data is *data debugging* which allows us to find incorrect data [MGNS11, MBM13]. Data may come from different, even heterogeneous sources, with ways unknown to the final user and thus often includes errors. How to find in which part of the data these errors exist, is an essential problem for modern companies.

Lineage records from where data comes from. For select-project-join SQL queries, lineage stores the set of *all* tuples that were used to compute a tuple in the answer of the query [BSH+08a]. This is natural for select-project-join SQL queries where original attribute values are “copied” in attribute values of the answer. However, in an aggregate query the value of the answer is the result of applying an aggregate function over many numerical attribute values. When we want to understand why we get an aggregate answer it may no longer be important or feasible to have lineage to point to all contributing original tuples and their values. We would rather want to compute *few* values that can be used to tell us as much as possible about the origin of the result of an aggregate query.

Storing the complete lineage can be very expensive, especially when dealing with big data. We study *efficient query computing* for *aggregate queries* and specifically for SUM queries posed on *big data*. We show that we can compute a suitable small useful lineage, called *Aggregate Lineage* that can be used, instead of the initial big data, for the answer of any aggregate SUM query, even when we do not know the query when we compute the small Aggregate Lineage. For example, we show that for an initial big dataset with one million tuples, we can compute a small Aggregate Lineage with size less than nine thousand tuples. It is important to note that an Aggregate Lineage can be computed once and be suitable for all possible SUM queries of any possible subset of the initial big data. Aggregate Lineage is computed without having knowledge of the SUM queries that are going to be posed. Specifically we present an Algorithm *Comp-Lineage* that computes in polynomial time a small Aggregate Lineage whose size is independent of the size of the initial big data. We also prove that this small Aggregate Lineage can be used to approximate well any SUM query and with time complexity depending only with its small size and hence in time independent of the initial big-sized data. As an example, when the initial data has 10^6 tuples an Aggregate Lineage with only $< 9,000$ tuples can be used to approximate well (with relevant error less than 10%) and with very high probability (i.e., error probability less than 10^{-6}) any sum that can involve any subset of the initial big dataset, if the sum in question is large (larger than the 0.4 of the total sum). We show that Aggregate Lineage includes the most possible information about initial data, given its small size, and that this information can be used in the important application of *data debugging*. For example, it can indicate why the answer of a SUM query is very large, larger than expected. These indications are, for example, approximately which of the initial data are responsible for this very large answer, even though we are only using the small Aggregate Lineage and not the initial big data.

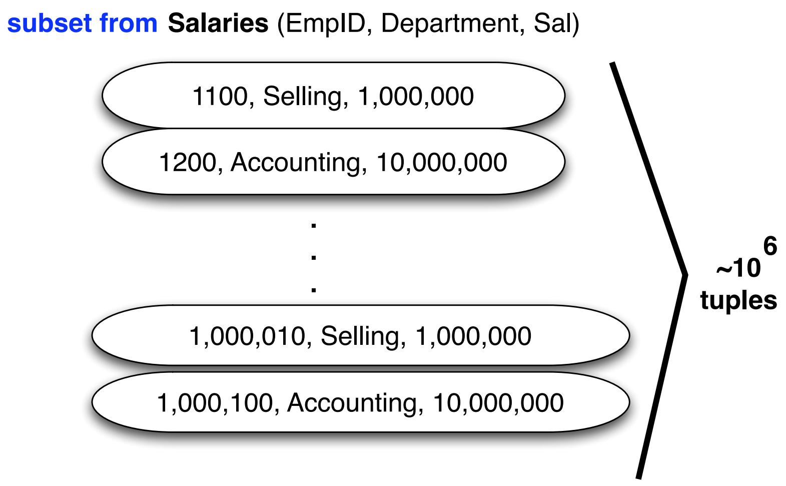

subset from Salaries (EmpID, Department, Sal)
1100, Selling, 1,000,000
1200, Accounting, 10,000,000
.
.
.
1,000,010, Selling, 1,000,000
1,000,100, Accounting, 10,000,000
~10 6
tuples

Figure 1.11: $SUM(Sal) \approx 10^{12}$. Infeasable to have lineage λ pointing to $\approx 10^6$ tuples.

Example 3. Suppose that the accounting department of a big company maintains a database with a relation *Salaries* with $\approx 10^6$ tuples. Each tuple in the relation contains an identifier of an employee stored in attribute $EmplID$, his Department stored in attribute *Department* and his annual salary stored in attribute *Sal*. A subset of *Salaries* relation is shown on Figure 1.11. If we query for the SUM of all *Sal* values, then the lineage of the answer would point to all $\approx 10^6$ tuples. In addition if we wanted to be able to have lineage that explains answers of all possible, even unknown, queries over the data then a naive approach would be to also keep all data. This approach of keeping the complete lineage is inefficient and can become even infeasable when dealing with big data.

For aggregate queries we would rather want to keep few representative values which can be used to tell us as much as possible (approximately) about the origin of an aggregate query answer. Suppose that we want to understand why we get an unexpectedly large answer to an aggregate SUM query. If the unexpectedly large answer is a result of tuples with few "very large" values and some "small" ones in the aggregated attribute then keeping only the few tuples with the "very large values" can provide the necessary information to understand why we get this large answer. On the other hand, if the unexpectedly large answer was a result of too many small values, we would like to be able to say that this large sum was due to the summing of many small values (and approximately which such values and how many).

Going back to our example, other relations can be extracted from the relation *Salaries*, e.g., a relation which contains aggregated data such as the total sum of salaries of all employees. A user is trying to use the second relation for decision making but he finds that the total sum of salaries is unacceptably high. He does not have easy access to the original relation or he does not want to waste time to pose time-consuming queries on the original big relation. The error could be caused by several reasons (duplication of data in a certain time period, incorrect code that computes salaries in a new department). Thus, if for example we could find the total sum of salaries for employees in the toy department, and see that this is unreasonably high, still close to the first total sum of all employees' salaries, then we will be able to detect such errors and narrow them down to small (and controllable) pieces of data.

We propose to keep as Aggregate Lineage a small relation under the same schema of the original relation. In order to select which tuples to include, we use valued-based sampling with repetition, i.e., weighted random sampling where the probability of selecting each tuple is proportional to its value on the summed attribute. The intuition why this method works is the following: Larger values contribute more to the sum than smaller ones, thus we expect that tuples with larger values should be selected more often than tuples with smaller values. Hence, we could end up with a tuple selected many times in the sample even if it appears only once in the original data. On the other hand, if there are many tuples with values of moderate size, many of them will be selected in the Aggregate Lineage, so that their total contribution to the approximation of the sum remains significant.

We illustrate the properties of Aggregate Lineage in our example. Suppose, for the sake of representation, that our initial big relation *Salaries* has tuples that can be divided into five groups according to their salary values: e.g. 100 employees have salary 10^9, 10^3 employees have salary 10^8, etc. This salary distribution is shown on Figure 1.12. As we said each tuple can be selected many times in Aggregate Lineage and the higher the salary value, the more probable it is for this tuple to be selected. We compute an Aggregate Lineage with only at most $8,852$ tuples out of the original $\approx 10^6$ - we give more details on Chapter 6. On Figure 1.13 we give the probabilities of selection in Lineage for the original tuples in the five salary groups. In Figure 1.14 we see how many times each tuple is selected: For example the original 100 tuples with salary values 10^9 are selected in the Aggregate Lineage 681 times (each tuple is selected many times). Thus we have in the Aggregate Lineage 100 tuples that correspond to the 100 original tuples with salaries 10^9. Each of these 100 tuples is selected in Aggregate Lineage many times (specifically $3-11$ times). This value of how many times a tuple is selected is represented in a new attribute Fr (for *Frequency*) in the Aggregate Lineage. Figure 1.15 has the details for this first group: We have 5 tuples in Aggregate Lineage that were selected 3 times, so with frequency 3. We also have 10 tuples with frequency 4, etc. In total we have from the first group 100 tuples in the Aggregate Lineage with frequencies $3-11$ that correspond to the 681 times that tuples from this group were selected with repetition $(3\cdot 5+4\cdot 10+5\cdot 19+6\cdot 14+7\cdot 13+8\cdot 15+9\cdot 8+10\cdot 12+11\cdot 4 = 681)$. As we see in Figure 1.14, Aggregate Lineage selected 100 out of the 100 original tuples from the first group, 497 out of the $1,000$ from the second group of tuples with original salary values 10^8, 681 tuples out of the $10,000$ original tuples from the third group and only $6,809$ tuples out of the 10^6 original tuples from the fourth group, while selects none of the 10 tuples with low salary 10^3. Hence, Aggregate Lineage has $< 9,000$ tuples out of the original $\approx 10^6$. Nevertheless this small Lineage can approximate well and explain all possible large SUM queries. Aggregate Lineage is a new relation with the same set of Attributes like the original $Salaries$ relation plus a new attribute Fr which stores the frequencies of selection. To illustrate its creation, in Figure 1.16 we suppose that the tuple with identifier 32 of is selected 3 times in Aggregate Lineage.

How can we use the Aggregate Lineage in order to approximate well large sums and to debug large answers? On Figure 1.17 we see the properties of the final Aggregate Lineage. The reason Aggregate Lineage can be used to approximate well large sums is the following: We can "intelligently" use the frequency of each tuple of the Lineage to compute a suitable approximated answer for the salary. Note that out of the $\approx 10^6$ original tuples, the Aggregate Lineage of our example selects only $8,087$. So we have to suitably "increase" the approximated salary value of some of the selected tuples in the Aggregate Lineage in order to still be able to approximate large sums out of it. The approximated salary value Sal' is given from the equation: $Sal' = Fr \cdot S/b$, where Fr is the frequency of a tuple in the Aggregate Lineage, S is the total sum and b is an upper bound of the tuples that the Aggregate Lineage can have (in our case we have $b = 8,852$ - we give more details on Chapter 6).

Tuples with original large salary values, i.e., 10^9 are selected with a frequency > 1 (c.f. Figure 1.17). Their approximated salary value that is computed from Aggregate Lineage in order to approximate sums, ranges from $3\cdot S/b = 4.41 \times 10^8$ to $11\cdot S/b = 1.62 \times 10^9$. We see that their approximated value is close to the original large one.

Now if we move to the group of the 10^6original tuples with salary values 10^6, we have that only $6,809$ of them are selected in Aggregate Lineage, all with frequency 1. The approximated salary value for them that is computed

Number of Employees	Salary
10^2	10^9
10^3	10^8
10^4	10^7
10^6	10^6
10^3	10

Figure 1.12: Salary distribution of our example.

from Aggregate Lineage in order to approximate sums, is given from $1 \cdot S/b = 1,47 \times 10^8$. We see that their approximated value is increased from their original value of 10^6 to $1,47 \times 10^8$.

Suppose that we have a SUM query Q_1 asking the sum of the salaries of a subset of the employees of the company defined from a subset of $EmpID$'s. Let this subset consist of 50 employees with salary 10^9, $5,000$ employees with salary 10^7 (so half of them) and of all 10^6 employees with salary 10^6. We compute the query over $Salaries$ and take the exact answer $Q_1 = 1.1 \times 10^{12}$.

The sub-lineage of Q_1, that is the lineage of Q_1 when is posed on the small Aggregate Lineage, points to 50 of the tuples of the Aggregate Lineage that correspond to original tuples with salaries 10^9 and to all $6,809$ tuples with original Sal values 10^6. It will also point to some tuples of the Lineage that had original Sal values 10^7: On average query Q_1 is applied on half of the 681 selected in Aggregate Lineage tuples, but in extreme cases it may include all or none of them.

In one worst case query Q_1 will include: the 50 tuples with original salaries 10^9 from Aggregate Lineage tuples with the larger frequencies and all 681 selected tuples with original salaries 10^7. The approximation Q_1' in this case is $(4 \cdot 11 + 12 \cdot 10 + \ldots + 681 + 6,809)S/b = 7,935 \cdot S/b = 1.17 \cdot 10^{12}$. In the other extreme case Q_1 includes tuples with the smaller frequencies and none of the selected in Aggregate Lineage tuples with salaries 10^7, yielding the approximation $6,995 \cdot S/b = 1.03 \cdot 10^{12}$. We see that Q_1 is well approximated. Of course the approximation bounds are not the same for every SUM query - we present the guarantees on Chapter 6.

Number of Employees	Salary	Probabilities
10^2	10^9	$7.69 \cdot 10^{-4}$
10^3	10^8	$7.69 \cdot 10^{-5}$
10^4	10^7	$7.69 \cdot 10^{-6}$
10^6	10^6	$7.69 \cdot 10^{-7}$
10^3	10	$7.69 \cdot 10^{-12}$

Figure 1.13: The probabilities of selecting tuples.

# tuples in *Salaries*	Salary	Probabilities	Selected # times	# tuples in Lineage
100	10^9	$7.69 \cdot 10^{-4}$	681	100
$1,000$	10^8	$7.69 \cdot 10^{-5}$	681	497
$10,000$	10^7	$7.69 \cdot 10^{-6}$	681	681
10^6	10^6	$7.69 \cdot 10^{-7}$	$6,809$	$6,809$
10^3	10	$7.69 \cdot 10^{-12}$	0	0

Figure 1.14: How many times each tuple is selected and the number of tuples in the Aggregate Lineage from each group.

# tuples in *Salaries*	Salary	Probabilities	Selected # times	# tuples in Lineage
100	10^9	$7.69 \cdot 10^{-4}$	681	100

# of Tuples in Lineage	Frequencies Fr in Lineage	# of Tuples with this Fr in Lineage
100	3	5
	4	10
	5	19
	6	14
	7	13
	8	15
	9	8
	10	12
	11	4

Figure 1.15: The first group with 100 tuples with original salaries 10^9 and how it is represented in Aggregate Lineage.

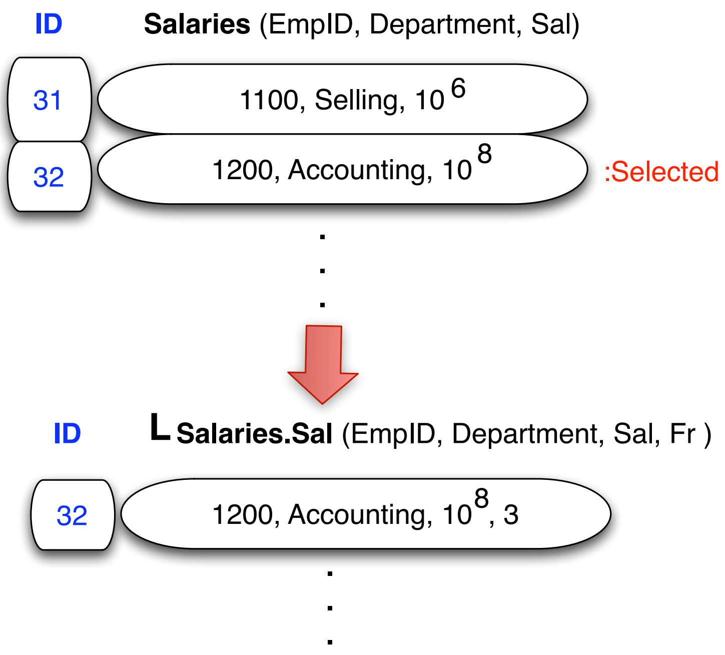
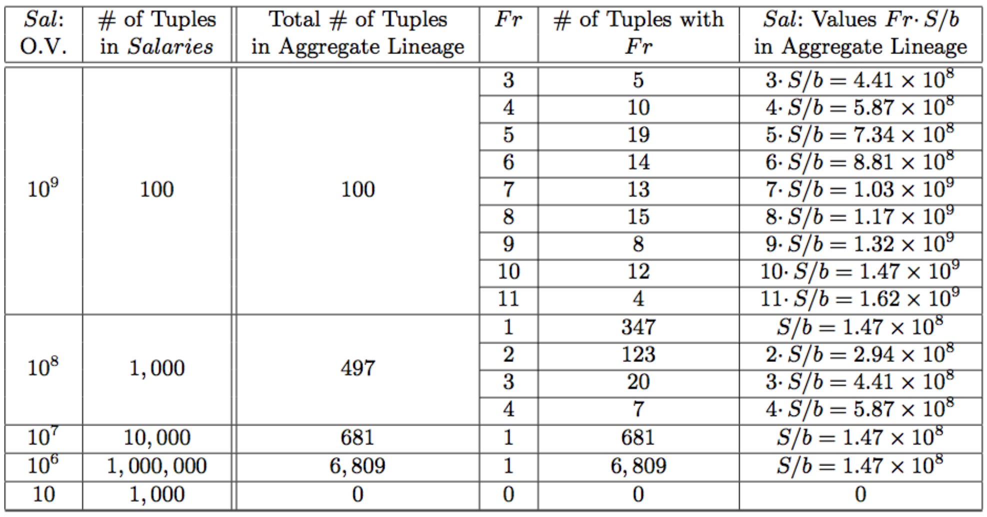

ID
Salaries (EmpID, Department, Sal)
31
1100, Selling, 10^6
32
1200, Accounting, 10^8
:Selected
ID
L **Salaries.Sal** (EmpID, Department, Sal, Fr)
32
1200, Accounting, 10^8, 3

Figure 1.16: From original data to Aggregate Lineage.

1.5 Optimising 2-Way Joins with Skewed Big Data on MapReduce

When dealing with big data, one approach to make efficient computations is to use the parallel environment *MapReduce*. The main challenges on this environment are: i) to define algorithms suitable for MapReduce parallelism paradigm and ii) handle efficiency that is critically reduced when we have data skew. In such a case a single node may be responsible for the biggest part of the total computation and this can significantly slow down the total computation, making it inefficient and close to the time needed if we had no parallelism at all. We present a new algorithm that is suitable for MapReduce and computes a 2-way Join and can efficiently handle data skewness in contrast with existing algorithms. We further prove that our algorithm has the least possible communication cost. Finally we implement the algorithm in Java language using MapReduce's open source project Hadoop. Our experiments match the theoretical ones about the minimisation of the communication cost.



Sal: O.V.	# of Tuples in *Salaries*	Total # of Tuples in Aggregate Lineage	*Fr*	# of Tuples with *Fr*	*Sal*: Values $Fr \cdot S/b$ in Aggregate Lineage
10^9	100	100	3	5	$3 \cdot S/b = 4.41 \times 10^8$
			4	10	$4 \cdot S/b = 5.87 \times 10^8$
			5	19	$5 \cdot S/b = 7.34 \times 10^8$
			6	14	$6 \cdot S/b = 8.81 \times 10^8$
			7	13	$7 \cdot S/b = 1.03 \times 10^9$
			8	15	$8 \cdot S/b = 1.17 \times 10^9$
			9	8	$9 \cdot S/b = 1.32 \times 10^9$
			10	12	$10 \cdot S/b = 1.47 \times 10^9$
			11	4	$11 \cdot S/b = 1.62 \times 10^9$
10^8	$1,000$	497	1	347	$S/b = 1.47 \times 10^8$
			2	123	$2 \cdot S/b = 2.94 \times 10^8$
			3	20	$3 \cdot S/b = 4.41 \times 10^8$
			4	7	$4 \cdot S/b = 5.87 \times 10^8$
10^7	$10,000$	681	1	681	$S/b = 1.47 \times 10^8$
10^6	$1,000,000$	$6,809$	1	$6,809$	$S/b = 1.47 \times 10^8$
10	$1,000$	0	0	0	0

Figure 1.17: Properties of an Aggregate Lineage with $8,087$ tuples. The first two columns describe the data. The next three columns describe the Aggregate Lineage relation. The last column shows how we use this lineage to compute sub-sums.

1.6 Summary of the thesis

The main contributions of this thesis are:

- Chapter 2: The preliminaries of our work are given, which include the technical details of the ULDB model for databases with Uncertainty and Lineage. This model is initially defined in [BSH+08a].
- Chapter 3 [AV10b]: Study and analysis of the *query containment* and *query equivalence* problems for databases with uncertainty and lineage:
  - Presentation and definition of five new kinds of semantics for *database containment* with uncertainty and lineage.
  - Proof that the complexity of *query containment* for databases with uncertainty and lineage and for *conjunctive queries (CQs)* remains *NP-complete* as it is with ordinary databases for all five new kinds of containment semantics.
  - Proof that the complexity of *query containment* for databases with uncertainty and lineage and for *Unions of conjunctive queries (UCQs)* is also *NP-complete.*

  - Proof that the complexity of *query equivalence* for databases with uncertainty and lineage for CQs and UCQs is also *NP-complete* for the first two kinds of ULDB database containment and *Graph-Isomorphism-complete* for the last three.
  - Finally we define five new “equality” semantics of ULDB containment and we show that are important for ULDB data integration purposes. The complexity of checking conjunctive query containment under all these five kinds of equality ULDB containment is NP-complete.

- Chapter 4 [AV10a]: Study and analysis of the *data exchange* problem for databases with uncertainty and lineage:
  - Presentation and definition of new logical semantics of *certain answers* for data with uncertainty and lineage.
  - Presentation of a new *u-chase* algorithm that extends the known *chase* algorithm which is about ordinary databases.
  - Proof that the new *u-chase* algorithm can be used for *query answering* for conjunctive queries with the complexity of computing *certain answers* that remains low, i.e. polynomial time (Ptime), as with ordinary databases when the set of dependencies between the source and the target schema is a *weakly cyclic* set of *tuple generating dependencies- tgds*.
  - Proof that when we also have target (equality generating dependencies - egds) then the problem of *query answering* for databases with uncertainty and lineage becomes *NP-hard* (in contrast with ordinary databases).

- Chapter 5 [VK11]: Study and analysis of the problem of query computing for conjunctive queries for databases with uncertainty and lineage when we attach belief values to uncertain data that come from a *possibility distribution*:
  - Proof that the model of databases with uncertainty and lineage and with possibilistic values is *closed* for conjunctive queries. This result solves the problem of previous results that show that models with uncertainty and possibilities (but no lineage) are not closed for conjunctive queries [BP05].
  - Proof that in the model of databases with uncertainty and lineage and possibilities we can compute conjunctive queries (along with

their possibility values) with low polynomial complexity (Ptime) in contrast with the high complexity #P of existing approaches where uncertainty values are probabilistic [DS04, GT06, RPT11, STW08].

- Chapter 6 [AFV14]: Study and analysis of the problem of *efficient* computing *aggregate queries* and specifically SUM queries, posed on *big data* using a small useful lineage instead of the initial big data:
  - Presentation of a new Algorithm *Comp-Lineage* which computes in polynomial time an *Aggregate Lineage* with small size that is independent of the size of the initial big data.
  - Proof that this small *Aggregate Lineage* can be used to approximate well any SUM query whose value is large, with time complexity that depends only on the small size of the *Aggregate Lineage* and hence independent with the large size of the initial big data.
- Chapter 7 [AUV]: Study, analysis and implementation in the parallel environment *MapReduce* of computing 2-Way Joins for *big data* that may also be *skewed*:
  - Presentation of a new algorithm, suitable for MapReduce, which computes 2-Way Joins and can efficiently handle *data skewness* in contrast with existing algorithms.
  - Proof that our algorithm matches the lower communication cost bound.
  - Implementation in Java/Hadoop of our algorithm with experiments whose communication cost verifies the theoretical cost.

Chapter 2

Preliminaries: The ULDB Model for Databases with Uncertainty and Lineage

In this Chapter we give the preliminaries of our work , which include the technical details of the ULDB model for databases with Uncertainty and Lineage. This model is initially defined in [BSH⁺08a].

2.1 Databases with Lineage -LDBs

LDBs extend the relational model in the way that apart from a set of relations $\bar{R}$ they also consist of a set of identifier symbols S and of lineage λ. In detail, lineage points to other tuples from which a tuple has been derived. To enable such pointing, tuples in LDBs come with an identifier. So we attach to each tuple in every relation a unique identifier. The set $I(\bar{R})$ contains all identifiers of relations $\bar{R}$. The definition of a database with lineage (LDB) is now the following:

Definition 1 (Database with lineage LDB)**.** [BSH⁺08a] An LDB D is a triple $(\bar{R}, S, \lambda)$, where: $\bar{R}$ is a set of relations $R_1, \ldots, R_n$. Each R_i is a multiset (bag) of tuples. We attach a unique identifier $ID(t)$ to each tuple t in the database, and $I(\bar{R})$ denotes all identifiers in relations $R_1, \ldots, R_n$. The set of symbols containing $I(\bar{R})$ and any possible external identifiers is S. With λ we denote the lineage function from S to 2^S.

Note that the lineage λ is a function from the set S of identifier symbols to the powerset of S. In general a tuple t^{LDB} in an LDB relation consists of: its identifier symbol, its data t and its lineage. We have the following definition:

Definition 2 (LDB tuple)**.** A tuple t^{LDB} of an LDB $D = (\bar{R}, S, \lambda)$ consists of three things: i) its data t which belongs to a relation in the set $\bar{R}$ of relations, ii) its unique identifier symbol denoted as $ID(t)$ and belonging to the set S of symbols and iii) its lineage $\lambda(ID(t))$ (belonging to λ) which associates it with the set of the identifiers of tuples from which it is derived. Thus t^{LDB} is a triple $\big(ID(t),\ t,\ \lambda(ID(t))\big)$.

When a tuple's identifier is clear from the context, we may abuse the above notation and refer to an LDB tuple only with its identifier ID (i.e., denote its lineage as $\lambda(ID)$). Alternatively, we may say that we want to have an LDB tuple with data t and lineage $\lambda(t)$ present in our database, when confusion does not arise. We refer to the data part of an *LDB tuple* t^{LDB} simply with the term *tuple* t of an LDB.

We now give the definition of the widely used class of Conjunctive Queries, which covers the SELECT,PROJECT,JOIN subset of SQL:

Definition 3 (Conjunctive Query - CQ)**.** A *conjunctive query* (CQ) over a schema $\bar{R}$ is an expression of the form $Q(\bar{x})$:-$\phi(\bar{x}, \bar{y})$, where $\bar{x}$ and $\bar{y}$ are sets of constants or variables. $Q(\bar{x})$ is called the *head* and $\phi(\bar{x}, \bar{y})$ is called the *body* of the query. An atomic formula has the form $R_m(t_1, \ldots, t_n)$ where R_m is a relation symbol of a relation with arity n and each t_i is a variable or a constant. The body $\phi(\bar{x}, \bar{y})$ is a conjunction of atomic formulas which are also called *subgoals* of the query. Duplicate atoms in the body of the query are removed.

Unlike traditional databases, the result $Q(D)$ of computing a query Q on an LDB D includes the original LDB relations *along with* the new relation that is the answer to the query. The reason for including original relations is the fact that the tuples in the query answer have lineage pointing back to them. Also in the answer of a query posed over an LDB we can have two alternatives with the same data if they have different lineage (so we can have a bag of tuples). We have the following algorithm:

Algorithm 1 (Computing CQs over LDBs) [AV10a, BSH$^+$08a]. Let $D = \{\bar{R}, S, \lambda\}$ be an LDB instance and Q a conjunctive query. A homomorphism h is a mapping from constants and variables to constants and variables such that each constance c is mapped to itself, i.e. $h(c) = c$. The answer of query Q posed on D is the LDB database $Q(D)$ that we get when we take all the LDB relations of D and add to them an LDB relation R_Q which will have:
a) tuples that are created when there exists a homomorphism h that maps variables occurring in Q to constants such that the following three hold:

1. h maps all the variables of Q to constants occurring in the tuples of D.
2. If $R_i(\bar{x}, \bar{y})$ is an atomic formula of the body of Q then $R_i(h(\bar{x}),\ h(\bar{y})\,)$ is a tuple of D.
3. If the head of Q is $Q(\bar{x})$, then the tuples in relation R_Q will have data $t = h(\bar{x})$ with data that we get from the head of Q when we substitute constants for variables in the body of Q and require that all subgoals become a tuple of D, and
b) lineage the union of the identifiers of all LDB tuples that are the images under h of the subgoals of the body of Q in step a.
We attach to each created tuple in R_Q a new unique identifier. The set of identifiers in R_Q is denoted with $I(R_Q)$.

To illustrate the above algorithm an example of computing a conjunctive query over an LDB:

Example 4. Let us consider the second possible instance of ULDB U of example 1 which is shown in Figure 1.3 on Chapter 1. Since D_2 is a possible instance it is an LDB. Consider again conjunctive query $Q_1(x) : -Drives(x, y)$, $Saw(z, y)$. The result $Q_1(D_2)$ of posing Q_1 over LDB Database D_2 is shown in Figure 2.1.

We note that in Definition 1 an LDB relation is a multiset (bag) of tuples. In Figure 2.1 LDB relation R_{Q1} has tuple *John* twice. This is semantically correct because *John* with identifier $(31, 2)$ comes from tuples $(John, Mazda)$ (with identifier $(11, 2)$) and $(Cathy, Mazda)$ (with identifier $(21, 2)$), while *John* with identifier $(32, 1)$ comes from tuples with identifiers $(11, 2)$ and $(22, 1)$. This difference in the origin is stored in the different lineage of tuples $(31, 2)$ and $(32, 1)$. As a result it is natural to allow duplicate tuples in LDBs if they have different lineage: now an LDB “remembers” where a tuple comes from and distinguishes two tuples with same data if they resulted from different tuples. In contrast a database with no lineage would only contain one tuple with data *John* in the answer of Q_1. Intuitively query Q_1 would mean for ordinary databases: return all the suspect names resulting from tuples in *Saw* and *Drives*. In contrast for LDBs query Q_1 means: return all the suspect names resulting from *different sets of tuples* of *Saw* and *Drives*.

2.2 Databases with Uncertainty and Lineage - ULDBs

A ULDB is an LDB with uncertainty. So again it will contain a lineage function λ and identifier symbols belonging to a set S. The difference is that

ID	**Drives**(name,car)
11,2	John,Mazda
12,1	Kate,Honda
13,1	Kate,Toyota

ID	**Saw**(witness,car)
21,2	Cathy,Mazda
22,1	Amy,Mazda

$\lambda(21,2) = \{(11,2)\}$

ID	R_{Q_1}
31,2	John
32,1	John

$\lambda(31,2) = \{(11,2),(21,2)\}$
$\lambda(32,1) = \{(11,2),(22,1)\}$

Figure 2.1: LDB $Q_1(D_2)$: Result of posing Q_1 over LDB D_2.

instead of ordinary tuples it will consist of x-tuples and its identifiers will also refer to alternatives. Uncertainty is encoded in these alternatives: for each x-tuple only one of them (or none if we have a '?' symbol) can be true in a possible instance, i.e., we have mutual exclusion between alternatives of a same x-tuple. We have the following definition for ULDBs:

Definition 4 (ULDB)**.** [BSH+08a] A ULDB D is a triple $D = (\bar{R}, S, \lambda)$, where $\bar{R}$ is a set of relations that consist of x-tuples which are a multiset (bag) of tuples called alternatives. The set S contains:
i) x-tuple identifiers which are unique numbers.
ii) alternative identifiers which are pairs of the form (i, j) where i is an x-tuple identifier and j points to the j-th alternative of x-tuple i, and
iii) possible external x-tuple and alternative identifiers. The lineage λ is a function from S to 2^S. Each x-tuple can also be annotated with a '?' symbol which denotes that there exist a possible instance which contains none of the alternatives of this x-tuple. Such an x-tuple is called a "maybe" x-tuple.

We call all alternatives that have empty lineage as *base data*. The *base lineage* of an alternative will include all the base alternatives that exist in its unfolded lineage back to base data:

Definition 5 (Base data, Base lineage, Transitive closure of lineage)**.** Let $U = (R, S, \lambda)$ be a ULDB. We refer to all alternatives in S that have empty lineage as "base data". Since the lineage $\lambda(t)$ of an alternative with data t

refers to other alternatives, we can expand the lineage of each alternative occurring in $\lambda(t)$ back to some base alternatives. With $\lambda^*(t)$ we denote the *transitive closure* of lineage, i.e., the set of all identifiers that we encounter when we expand $\lambda(t)$ until we reach a set of base data identifiers. We call *base lineage*, denoted with $\lambda_B(t)$, the subset of the unfolded lineage of an alternative with data t that contains only base alternative identifiers.

To illustrate the above notions let us suppose that we pose query $SuspectsInformation(name, car)$:-$Drives(name, car), Saw(witness, car)$ on our running ULDB U of example 1 on Chapter 1. The result will be a ULDB U along with relation $R_{SuspectsInformation}$ in which there exists one alternative with data $(John, Honda)$ coming from alternatives $(11, 1)$ and $(21, 1)$. Let its ID be $(31, 1)$. Suppose that over this result we then pose query $SuspectsCars(car)$:-$SuspectsInformation(name, car)$. In new relation $R_{SuspectsCars}$ there exists one alternative with data $(Honda)$ coming from alternatives $(31, 1)$. Let its ID be $(41, 1)$. The lineage of alternative $(41, 1)$ in $R_{SuspectsCars}$ is $\lambda(41, 1) = \{(31, 1)\}$. Alternative $(31, 1)$ is not base, so we can expand $(31, 1)$ to its lineage in $R_{SuspectsInformation}$, so to $\{(11, 1), (21, 1)\}$. Now $(21, 1)$ points to base $(11, 1)$. Hence we have that the transitive closure of $\lambda(41, 1)$ is $\lambda^*(41, 1) = \{(31, 1), (11, 1), (21, 1)\}$. The base lineage of $(41, 1)$ is $\lambda_B(41, 1) = \{(11, 1)\}$.

We give the semantics of a ULDB as representing a set of possible instances, where each instance is an LDB:

Definition 6 (Possible instances)**.** [BSH+08a] Let $D = (\bar{R}, S, \lambda)$ be a ULDB. A possible LDB D_k of D is obtained as follows. Pick a set of symbols $S_k \subseteq S$ such that:

- If $(i, j) \in S_k$, then for every $j' \neq j$, $(i, j') \notin S_k$.
- $\forall (i, j) \in S_k$, $\lambda(i, j) \subseteq S_k$.
- For any t_i such that there does not exist a $(i, j) \in S_k$, the following hold: (i) t_i is a maybe x-tuple, and (ii) $\forall (i, j) \in t_i$, either $\lambda(i, j) = \emptyset$ or $\lambda(i, j) \not\subseteq S_k$.

The possible LDB D_k is the triple $(\bar{R}_k, S_k, \lambda_k)$ where $\bar{R}_k$ includes exactly the alternatives of x-tuples in $\bar{R}$ such that $(i, j) \in S_k$, and λ_k is the restriction of λ to S_k.

Intuitively, the first condition in Definition 6 captures the semantics of alternatives of a same x-tuple: they are mutually exclusive and thus only one of them can appear in a possible instance. The second condition concerns lineage semantics. If an alternative is present in a possible instance, so must be the facts from which it was derived. The third condition says that one of the alternatives of an x-tuple must be selected and appear in a possible instance unless: (i) x-tuple has an '?' *and* (ii) none of its alternatives has (nonempty) lineage that would make it to appear due to condition 2 (in this case '?' will be selected for this maybe x-tuple). Figures 1.2 and 1.3 on Chapter 1 show the two possible instances of ULDB U of Example 1.

In our definitions so far the lineage of a ULDB can be arbitrary and nonintuitive. Consider for example the ULDB of Figure 2.2. According to Definition 6 one possible instance is the one that selects alternatives $(11, 1)$ and $(21, 1)$. A second possible instance selects again $(11, 1)$ and this time selects $(21, 2)$. The third possible instance is empty for both relations. The previous definition allows $(11, 1)$ to be selected in a possible instance without having both $(21, 1)$ and $(21, 2)$ (that have their lineage equal with $(11, 1)$) to appear in it. This is why the second condition implies that when the lineage of some alternatives is present in a possible instance we do not always have that all such alternatives will also be present. On the other hand if the lineage of some alternatives of a same x-tuple is present in a possible instance one of them will appear in it according to the mutual exclusiveness of alternatives, the semantics of '?' symbol and first, second conditions of Definition 6. This is why the only case that none of the alternatives of an x-tuple are present in a possible instance can only occur when it is a maybe x-tuple and the lineage of none of its alternatives appears in it.

Well-Behaved Lineage

The lineage of ULDB in Figure 2.2 is not intuitive: Two mutual exclusive alternatives are derived from a same fact. In practice lineage satisfies natural properties. In [BSH+08a] those properties are defined and lineage satisfying them is called "well-behaved":

Definition 7 (Well-Behaved Lineage)**.** [BSH+08a]
The lineage of an x-tuple t_i is well-behaved if it satisfies the following three conditions:
1. Acyclic: $\forall(i, j),\ \ (i, j) \notin \lambda^*(i, j)$.
2. Deterministic: $\forall(i, j), \forall(i, j')$ if $j \neq j'$ then either $\lambda(i, j) \neq \lambda(i, j')$ or $\lambda(i, j) = \emptyset$.
3. Uniform: $\forall(i, j), \forall(i, j')$ $B(i, j) = B(i, j')$ where $B(i, j) = \{t_k \mid \exists(k, l) \mid (k, l) \in \lambda(i, j)\}$.

As discussed it is natural for the lineage of two different alternatives of a same x-tuple to not be equal (condition 2 of well-behaved lineage Definition 7). Similarly in order for alternatives of an x-tuple to be mutually exclusive they must point to mutually exclusive sets of alternatives, so to alternatives from a same set of x-tuples (condition 3 of well-behaved lineage Definition

ID	**Drives**(name,car)
11	John,Honda

ID	**Saw**(witness,car)
21	Cathy,Honda || Cathy,Mazda

$$\lambda(21,1) = \{(11,1)\}$$
$$\lambda(21,2) = \{(11,1)\}$$

Figure 2.2: Non well-behaved ULDB U_1

7). Finally lineage is created to connect an alternative in an answer of a query to alternatives already existing in the database from which the answer was derived or to capture some logical relation between different alternatives. Hence condition 1 is also natural. Since alternatives of a same x-tuple cannot occur in a same possible LDB instance, we allow alternatives of a same x-tuple to have the same data (alternatives of an x-tuple are a bag of tuples) even if they have same empty lineage (from condition 2 of Definition 7 two different alternatives of a same x-tuple cannot have same lineage unless it is the empty lineage).

In [BSH+08a] the following was showed: If we start from a base ULDB with empty lineage or from a well-behaved ULDB and perform conjunctive queries (queries including select, project and join operations) creating the natural lineage for the results, the ULDB remains well-behaved. In [BSH+08a] it was also shown that if we add tuples created from conjunctive queries that have well-behaved lineage pointing to existing data then those tuples do not alter the possible instances of previously existing relations. Since we are interested in conjunctive query containment and practical kinds of ULDB containment we always assume that our ULDBs are well-behaved.

In addition a ULDB that represents a set of given possible LDBs (so tuples appearing in different possible given LDBs will never have lineage that make them appear in one possible instance of that ULDB) has also "well-behaved" lineage. Note that from condition 2 of the above definition we can no longer have two alternatives of a same x-tuple with the same non empty lineage and as a result for well-behaved ULDBs a non base alternative is present in a possible instance if and only if its lineage appears in it.

One of the desirable properties of a ULDB instance with "well-behaved" lineage is that its possible instances are determined entirely by the alternative choices of the base x-tuples. We have the following relevant Theorem which is proven again in [BSH+08a]:

Theorem 1 (Well-behaved instances)**.** *Let D_1 and D_2 be two possible instances of a well-behaved ULDB $D = (\bar{R}, S, \lambda)$. Then $D_1 = D_2$ if and only if D_1 and D_2 pick the exact same alternative or '?' for every base x-tuple.*

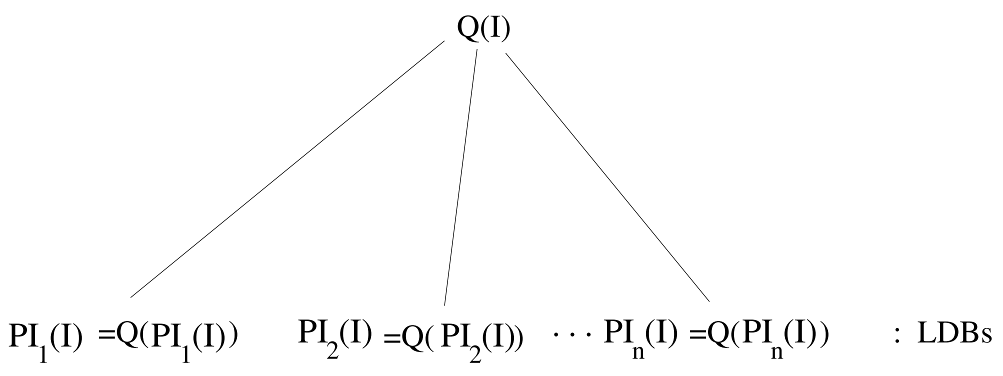


Figure 2.3: Semantics of posing a query Q over a ULDB I.

2.3 Computing Conjunctive Queries (CQs) on the ULDB model

Semantics of computing a CQ over a ULDB, shown in Figure 2.3, are defined in [BSH+08a]:

Definition 8 (Semantics of computing a CQ over ULDBs)**.** Let a CQ Query Q and a ULDB U. Suppose that U has n possible instances: $PI(U) = \{PI_1(U), PI_2(U), \ldots, PI_n(U)\}$. The result of applying Q on U is a ULDB $Q(U)$ that contains ULDB U and a new ULDB relation $R_q(I)$ such that:

ULDB $Q(U)$ has possible instances $\{PI'_1(U), PI'_2(U), \ldots, PI'_n(U)\}$, where each $PI'_j(U)$ is obtained after posing query Q on $PI_j(U)$.

However explicitly computing the possible instances of a ULDB U is computationally expensive: from Definition 6 the tuples of each $PI(U)$ must satisfy complex logical formulas of their lineage. In contrary in [BSH+08a] an algorithm that computes answers of queries for ULDBs in polynomial time was presented. In this algorithm computing a query is based on first transforming the ULDB to a "pseudo-LDB" and then apply the query to it. We present in Algorithm 2 a slight variation of this algorithm, suitable for conjunctive queries. We need first to give a formal definition of the LDB that is the result of transforming a ULDB to a "pseudo-LDB", which is called the "Horizontal Database" of the ULDB:

Definition 9 (Horizontal Database)**.** Let U be a ULDB with x-relations $\{R_1, \ldots, R_n\}$. We define the Horizontal database of U and denote it with U_H the "pseudo-LDB": $U_H = R_{1H}, \ldots, R_{nH}$ such that $\forall k$, $k \in [1, n]$:

$R_{kH} = \{$ tuples $(i, j) \mid (i, j)$ is an alternative in $R_k\}$

Intuitively in order to take the Horizontal database of a ULDB we "flatten" each alternative of an x-tuple so that it will become a new tuple.

Algorithm 2: Computing CQs on a ULDB [BSH+08a]
input: a ULDB U with x-relations $\{R_1, \ldots, R_n\}$,
and a Conjunctive Query Q on U

output: a ULDB $U' = Q(U)$

```
1. $R_q \leftarrow \emptyset$ ; $\lambda_{R_q} \leftarrow$ undefined function

2. From ULDB $U$ create the Horizontal database, "pseudo-LDB" $U_H = R_{1_H}, \ldots, R_{n_H}$. More specifically:
   For each alternative $(i, j)$ of ULDB $U$ create a new tuple in $U_H$ that will have the same data with alternative $(i, j)$ and with identifier the pair $(i, j)$ which represents $j$ alternative of x-tuple $i$ for original ULDB $U$.

3. Since $U_H$ is a "pseudo-LDB" we can now use Algorithm 1 to compute: $Q(U_H) = U_H + (R_{q_H}, I(R_{q_h}), \lambda(R_{q_H}))$

4. Now for each tuple in LDB relation $R_{q_H}$, create a ULDB relation $R_q$ by creating x-tuples in the following way: if two or more tuples of the horizontal relation have lineage pointing to the same set of x-tuples then make them alternatives of the same x-tuple in $R_q$. Add to each x-tuple of $R_q$ symbol '?' if there exist a combination of x-tuple alternatives that is not in the result of $R_q$. Retain the lineage relationship of $Q(U_H)$.

5. return $Q(U) = U + (R_q, I(R_q), \lambda_{R_q})$
```

We now give an example of the steps of Algorithm 2:

Example 5. Let us consider again ULDB U of example 1 and conjunctive query $Q_1(x) : -Drives(x, y), Saw(z, y)$. We see in Figures 2.4 and 2.5 the steps 2 and 3 of Algorithm 2 when it is run with U and Q_1 as its inputs and in Figure 2.6 its output.

The fact that the output of Algorithm 2 has the correct semantics and is well-behaved is proven in [BSH+08a]:

Theorem 2. *Given a ULDB U and a conjunctive query Q:*

1. *Algorithm 2 returns $Q(U)$ with correct semantics.*
2. *If U is a well-behaved ULDB, then so is $Q(U)$.*

ID	**Drives$_H$ (name,car)**
11,1	John,Honda
11,2	John,Mazda
12,1	Kate, Honda
13,1	Kate, Toyota

ID	**Saw$_H$ (witness,car)**
21,1	Cathy,Honda
21,2	Cathy,Mazda
22,1	Amy,Mazda

$$\lambda(21,1) = \{(11,1)\}$$
$$\lambda(21,2) = \{(11,2)\}$$

Figure 2.4: Computing CQs over a ULDB step 2: Horizontal Database.

ID	**Drives$_H$ (name,car)**
11,1	John,Honda
11,2	John,Mazda
12,1	Kate, Honda
13,1	Kate, Toyota

ID	**Saw$_H$ (witness,car)**
21,1	Cathy,Honda
21,2	Cathy,Mazda
22,1	Amy,Mazda

$$\lambda(21,1) = \{(11,1)\}$$
$$\lambda(21,2) = \{(11,2)\}$$

ID	R_{Q1H}(U)
31	John
32	John
33	John
34	Kate

$$\lambda(31) = \{(11,1),(21,1)\}$$
$$\lambda(32) = \{(11,2),(21,2)\}$$
$$\lambda(33) = \{(11,2),(22,1)\}$$
$$\lambda(34) = \{(12,1),(21,1)\}$$

Figure 2.5: Computing CQs over a ULDB step 3.

ID	**Drives**(name,car)
11	John,Honda || John,Mazda
12	Kate,Honda
13	Kate,Toyota

ID	**Saw**(witness,car)
21	Cathy,Honda || Cathy,Mazda
22	Amy,Mazda

$$\lambda(21,1) = \{(11,1)\}$$
$$\lambda(21,2) = \{(11,2)\}$$

ID	$R_{Q_1}(U)$
31	John || John
32	John '?'
33	Kate '?'

$$\lambda(31,1) = \{(11,1),(21,1)\}$$
$$\lambda(31,2) = \{(11,2),(21,2)\}$$
$$\lambda(32,1) = \{(11,2),(22,1)\}$$
$$\lambda(33,1) = \{(12,1),(21,1)\}$$

Figure 2.6: ULDB $Q_1(U)$: Result of posing Q_1 on U.

The containment tests for conjunctive query containment that we define in the next Chapter make use of the following definitions:

Definition 10 (Containment Mapping)**.** Let Q, Q' be two conjunctive queries, and let h be a mapping from variables and constants of Q' to variables and constants of Q such that h is the identity for constants. We say that h is a *containment mapping* from Q' to Q if $h(head(Q')) = head(Q)$ and every atom in the body of Q' is mapped to an atom of the body of Q with the same predicate.

Definition 11 ((Subgoal) Onto-Containment Mapping)**.** A containment mapping from Q' to Q is (subgoal) *onto* if we additionally have that the set of images of all the subgoals of Q' contains *every* subgoal of the body of Q.

Chapter 3

Query Containment and its Complexity for Databases with Uncertainty and Lineage

We define and investigate the computational complexity of the query containment problem for data that support both uncertainty and lineage. Query containment depends on the definition of database containment which, for traditional databases, is defined as a simple *set containment* for each relation. As this is not the case in the presence of uncertainty and lineage, first we revisit the notion of database containment and define various kinds of it that may be natural in different practical situations. We give examples that illustrate why each different kind might be more suitable than the others and study the exact interrelationship among them as concerns implication.

We investigate query containment under lineage and uncertainty for the various kinds of database containment that we introduce for conjunctive queries (CQs) and their unions (UCQs). Even though all semantics of database containment are different, it turns out that for conjunctive query containment the variants fall in two categories as concerns the containment test. We further study equivalence for both CQs and UCQs. We show that the complexity of CQ and UCQ containment is NP-complete under all different kinds of query containment that we introduce. For CQ and UCQ equivalence we prove that the complexity is NP-complete for the semantics of the first two semantics and Graph-Isomorphism-complete for the last three. Finally we define five new "equality" semantics of ULDB containment and we show that are important for ULDB data integration purposes. The complexity of checking conjunctive query containment under all these five kinds of equality ULDB containment is NP-complete.

3.1 Introduction

Uncertain data appears in many modern applications including information extraction from the web, bio-informatics, scientific databases, entity resolution and sensors. Those and many other applications also require keeping track of the derivation of data, called *provenance* or *lineage*. There has been a lot of recent research that considers systems managing data with uncertainty [AKG91, BGMP92, DS04, IL84, SUW09], systems managing data with lineage tracking [BKT01, BT07, CW00, CW03] and systems that combine data with uncertainty and lineage [BSH+08a]. Semantics and algorithms for computing queries have been defined in those systems. To the best of our knowledge the problem of query containment has not been considered for a database system that handles uncertain data and also supports lineage tracking. We are going to investigate different kinds of query containment for the ULDB (Uncertainty Lineage DataBase) data model used in Trio System [BSH+08a] that will be based on different semantics of database containment for this model.

The problem of query containment arises in many important database applications like query optimization [ALU07], data integration, query answering using views [AK10, ALM06, Ull97], data exchange [FKMP05] and data warehousing. One of the reasons that the ULDB model was introduced was because it would be important in data integration and data exchange settings [BSH+08a]. In addition query optimization is recognized as one of the important open problems of the Trio system. It is already known from ordinary databases that both these problems rely on query containment.

A database query Q defines a mapping from databases to databases. A query Q_1 is said to be contained in a query Q_2 if for every database D, database $Q_1(D)$ is contained in database $Q_2(D)$. For ordinary data- bases, a database D_1 is contained in a database D_2 if the tuples of every relation in D_1 are contained in the corresponding relation of D_2 *as a set*. A relation of an uncertain database however semantically is not a set; it represents a set of *possible instances* - *PIs* (that have no uncertainty). The answer of a query over an uncertain database is a new uncertain database. Thus, when moving to ULDBs or even to uncertain databases without lineage, the set containment between answers of queries no longer applies.

On the ULDB model a database consists of uncertain tuples. An uncertain tuple called *x-tuple* consists of a bag of tuples called *alternatives*. At most one of those alternatives can exist in a possible instance. If for an x-tuple we can select none of its alternative in a possible instance then this tuple is called maybe x-tuple and annotated with “?” symbol. In particular, apart from their data, ULDB alternatives also consist of: i) a unique identifier

ID	**Drives**(name,car)
11	John,Honda || John,Mazda
12	Kate,Honda
13	Kate,Toyota

ID	**Saw**(witness,car)
21	Cathy,Honda || Cathy,Mazda
22	Amy,Mazda

$$\lambda(21,1) = \{(11,1)\}$$
$$\lambda(21,2) = \{(11,2)\}$$

Figure 3.1: Running Example: ULDB U.

and ii) a lineage function that connects them to other alternatives through the set of their unique identifiers. An uncertain database with lineage represents a set possible instances that are databases with lineage (LDBs) and have no uncertainty. As a result, ULDB database containment should not only consider data containment and it should also be based on the possible instances of a ULDB. Query containment between queries Q_1 and Q_2 will be based on ULDB database containment between the ULDB relations that are the answers of the two queries. The fact that ordinary conjunctive query containment (with containment mapping as a test) does not always suffice for ULDBs (or LDBs) is illustrated in the following example:

Example 6. Consider an uncertain database with lineage U containing information about names of persons who drive cars, stored in relation `Drives (name, car)` and about names of witnesses that saw a car near a crime-scene, stored in relation `Saw(witness,car)`[1]. Data in `Drives` contain uncertainty (e.g., due to unclear writing). Suppose we have uncertainty whether $John$ drives a $Honda$ or a $Mazda$ car. Figure 3.1 depicts the two relations `Drives` and `Saw` of ULDB U. We have an x-tuple with two alternatives $(John, Honda)$ and $(John, Mazda)$, whereas we are sure that $Kate$ drives a $Honda$ and a $Toyota$ car. Data in relation `Saw` have also uncertainty. Let us also suppose that $Cathy$ stated that she saw $John$ driving. Thus if $John$ drives a $Honda$ then $Cathy$ would have seen a $Honda$ car. This connection is captured through lineage between alternatives $(21, 1)$ and $(11, 1)$ (similarly for $(21, 2)$ and $(11, 2)$). All other alternatives have empty lineage which is omitted.

ULDB U represents two possible LDB instances, D_1, shown in Figure 3.2 and D_2, shown in Figure 3.3. Consider the following two Conjunctive Queries (CQs): $Q_1(x)$:-$Drives(x, y), Saw(z, y)$ and $Q_2(x)$:-$Drives(x, y)$.

[1]The general setting of our example is similar to the running example found in [BSH+08a].

ID	**Drives**(name,car)
11,1	John,Honda
12,1	Kate,Honda
13,1	Kate,Toyota

ID	**Saw**(witness,car)
21,1	Cathy,Honda
22,1	Amy,Mazda

$\lambda(21,1) = \{(11,1)\}$

Figure 3.2: D_1: First Possible LDB Instance of ULDB U.

ID	**Drives**(name,car)
11,2	John,Mazda
12,1	Kate,Honda
13,1	Kate,Toyota

ID	**Saw**(witness,car)
21,2	Cathy,Mazda
22,1	Amy,Mazda

$\lambda(21,2) = \{(11,2)\}$

Figure 3.3: D_2: Second Possible LDB Instance of ULDB U.

For ordinary databases, query Q_1 is contained in query Q_2 because there exists a containment mapping from Q_2 to Q_1 (which maps variables x and y to themselves). Consider the second possible instance D_2 of ULDB U. The answer of Q_1 posed on D_2 is the LDB relation R_{Q1} shown in Figure 3.4 (along with relations `Drives` and `Saw`). Respectively Q_2 posed over D_2 is the LDB relation R_{Q2} shown in Figure 3.5. We see that if we care only about data we have that indeed data *John* in the answer of Q_1 is contained in the set of tuples in the answer of Q_2 ($\{John, Kate\}$). But LDBs (and ULDBs) also contain lineage: The LDB of Figure 3.4, contains the lineage information that tuple $(31, 2)$ with data *John* comes from two different tuples $(11, 2)$ and $(21, 2)$. In contrast in the LDB of Figure 3.5 the only tuple with data *John* comes from tuple $(11, 2)$. Thus we have to decide what kind of containment between lineage information we should have in order to define meaningful LDB and ULDB containment.

In this chapter we start our investigation with considering various kinds of ULDB database containment. For each type of ULDB database containment we define corresponding kinds of ULDB query containment, give conjunctive query containment test and study its complexity. We also investigate in which cases each kind of containment is more suitable. In Figure 3.6 we list the various kinds of database containment and show the relation between them. Even though the database semantics for those kinds of containment are different it will turn out that some of them are equivalent for conjunctive query containment. We will refer in Figure 3.6 in more detail in Section 3.7.

ID	R_{Q_1}
31,2	John
32,1	John

$$\lambda(31,2) = \{(11,2),(21,2)\}$$
$$\lambda(32,1) = \{(11,2),(22,1)\}$$

Figure 3.4: D_{Q12}: LDB relation R_{Q_1} for: $Q_1(x) : -Drives(x,y), Saw(z,y)$ posed on D_2.

ID	R_{Q_2}
41,2	John
42,1	Kate
43,1	Kate

$$\lambda(41,2) = \{(11,2)\}$$
$$\lambda(42,1) = \{(12,1)\}$$
$$\lambda(43,1) = \{(13,1)\}$$

Figure 3.5: D_{Q22}: LDB relation R_{Q_2} for: $Q_2(x) : -Drives(x,y)$ posed on D_2.

One obvious kind of ULDB containment is based on the LDB containment defined on Trio which requires the containment of lineage in the transitive closure of lineage of the containing relation (Semantics #3 in Figure 3.6). We show why this kind of LDB containment may be inappropriate for some cases. Hence we relax this definition and yield a new kind of ULDB containment based on LDB Data Containment that requires the set containment of data of every possible instance of a ULDB (Semantics #1 in Figure 3.6). Further we define two kinds of database containment that take into account only *base lineage* (lineage extended back and referring only to alternatives with empty lineage, called *base alternatives*). The first kind requires *contained* base lineage (Semantics #2) and the second (Semantics #4) requires *same* base lineage of the data that is contained between two databases. Semantics #4 was shown to be suitable for data exchange [AV10a]. The fifth database containment semantics will require the containment of all data and lineage, not only base (Semantics #5).

Query containment for ordinary databases is known to be NP-complete for conjunctive queries [CM77]. A ULDB represents a set of possible instances that are LDBs and whose number can be exponential to its size. In addition lineage imposes complex logical formulas to alternatives that can be true on each possible instance. Furthermore a possible instance is an LDB and contains a bag of tuples (if they have different lineage). Query containment under bag semantics for CQs is known to be Π_2^p-hard [ADG10, CV93]. So we would expect ULDB query containment to be harder than ordinary set or bag query containment. In contrast we show that ULDB query containment for CQs is NP-complete for all the different kinds of containment which we study. In Section 3.8 we study equivalence for CQs and in Section 3.9 containment and equivalence for Unions of CQs (UCQs). We show that for CQs and UCQs the complexity of query equivalence is NP-complete under Data and CBase semantics and Graph-Isomorphism-complete for the other three containment semantics, while UCQ query containment is NP-complete for all semantics. In [ASUW10] another notion of containment has been defined for uncertain databases without lineage. It was defined in order to be suitable for data integration purposes. We prove which conditions should hold for conjunctive query containment under this kind as well in Section 3.11. In the same section we also define five new "equality "semantics of ULDB containment which we show that are important for ULDB data integration purposes. We prove that the complexity of checking conjunctive query containment under all five kinds of equality ULDB containment is NP-complete.

Related Work

The need for defining database containment in the presence of uncertainty and/or lineage has been noticed in [ASUW10], [BSH$^+$08a]. For databases with lineage, containment has been defined in various works: In [BSH$^+$08a] for *LDBs* and in [Gre09] for various kinds of semirings. [Gre09] is the first work that studies the complexity of CQ and UCQ query containment for databases with lineage. Specifically CQ and UCQ containment, equivalence and their complexity was investigated for databases with semiring annotations. The semiring model captures many kinds of other provenance models one of which is the model of LDBs. For LDBs and Same-Lineage semantics Theorems 5, 7, 9 and 11 are essentially derived in [Gre09]. Finally in [LLRS97] query equivalence and containment is investigated for probabilistic data.

3.2 Running Example and the ULDB Data Model

In this section we present a motivating example illustrating the need of defining new kinds of database containment, suitable for query containment. Through this example we will also explain the Trio ULDB model whose

#	**Semantics**	**Features**	**Implies DB cont.**	**CQ Containment Test**
1	Data	Set Contained Data	–	Containment Mapping
2	CBase-Lineage	Contained Base Lineage	1	Containment Mapping
3	TR-Lineage	Contained Transitive Closure of Lineage	1	Onto Mapping
4	SBase-Lineage	Same Base Lineage	1,2	Onto Mapping
5	Same-Lineage	Same Lineage	1,2,3,4	Onto Mapping

Figure 3.6: Comparison of Different Semantics.

formal definitions will be given in the next section. An uncertain database with lineage (ULDB) represents a set of *possible instances* (PI) which are databases with lineage (LDBs). Suppose that we have the ULDB U and the two conjunctive queries Q_1 and Q_2 from our Example 6. In the ULDB model the uncertainty of the value of a tuple (i.e. whether $John$ drives a $Honda$ or a $Mazda$ car) is represented through x-tuples. In general an x-tuple is a bag of ordinary tuples which we call *alternatives* and we separate them with symbol '||'. The semantics of alternatives in x-tuples are that at most one of them can be true in a *possible instance*. If we can have a possible instance that selects none of the alternatives of an x-tuple, then we annotate this x-tuple with '?' symbol.

In order to succinctly represent lineage connections between alternatives we attach to each x-tuple a unique *identifier*. For example, in Figure 3.1, x-tuple $(John, Honda)||(John, Mazda)$ in `Drives` has identifier 11. If the identifier of an x-tuple is i, then we refer to its j-th alternative with an *alternative identifier* which will be a pair (i, j). We represent the lineage connection between alternatives $(Cathy, Honda)$ and $(John, Honda)$ with a lineage function λ that connects alternative identifiers to sets of alternative identifiers, e.g.,: $\lambda(21, 1) = \{(11, 1)\}$. *Base data* of a ULDB instance consists of all data that have empty lineage. If two alternatives point, maybe after many lineage steps, to the same set of base data we say that they have the same *base lineage*. In our example U has two possible instances: one for each possible alternative selection of x-tuple 11.

ID	R_{Q_1}
31	John || John
32	John '?'
33	Kate '?'

$$\lambda(31,1) = \{(11,1),(21,1)\}$$
$$\lambda(31,2) = \{(11,2),(21,2)\}$$
$$\lambda(32,1) = \{(11,2),(22,1)\}$$
$$\lambda(33,1) = \{(12,1),(21,1)\}$$

Figure 3.7: ULDB Relation R_{Q_1} for: $Q_1(x) : -Drives(x,y), Saw(z,y)$.

ID	R_{Q_2}
41	John || John
42	Kate
43	Kate

$$\lambda(41,1) = \{(11,1)\}$$
$$\lambda(41,2) = \{(11,2)\}$$
$$\lambda(42,1) = \{(12,1)\}$$
$$\lambda(43,1) = \{(13,1)\}$$

Figure 3.8: ULDB Relation R_{Q_2} for: $Q_2(x) : -Drives(x,y)$.

Suppose we perform a natural join of `Drives` and `Saw` over common attribute *car* and a projection of attribute *name* on the result. Conjunctive query $Q_1(x)$:- $Drives(x, y), Saw(z, y)$ performs this operation. Intuitively Q_1 will return the names of suspects, i.e., persons who drive a car that was seen near the crime-scene. The result of computing Q_1 over ULDB U is a new ULDB $Q_1(U)$ that includes ULDB relations `Saw` and `Drives` of U and a new ULDB relation R_{Q_1} which is shown in Figure 3.7. The semantics of $Q_1(U)$ are that its possible instances should be the same with the possible instances we would have if we first considered the possible instances of U and computed Q_1 over each one of them.

The possible instance D_2 of ULDB U is shown in Figure 3.3. The corresponding possible instance of ULDB $Q_1(U)$ is the LDB possible instance which includes relations of D_2 along with LDB relation R_{Q1} shown in Figure 3.4 which is one of the possible instances of ULDB relation R_{Q_1} shown in Figure 3.7. Note that a possible instance of a ULDB is an LDB, so it does not have uncertainty (no alternatives separated with $||$), but only ordinary tuples with unique tuple identifiers and lineage. Since we have no alternatives, LDB tuples are always present and their unique identifiers can be single numbers and not pairs. For uniformity in the LDB possible instances of our ULDBs we use alternative identifier pairs to identify a tuple.

3.2.1 Non TR-lineage query containment

As we mentioned we define first variants for ULDB database containment and then we base our variants for query containment on these definitions. It turns out that, for conjunctive queries our variants fall in only two classes, where the semantics in each class are shown to be CQ-containment equivalent, in that if two conjunctive queries are contained in each other according to one semantics in the class, then they are also contained according to the other semantics in the same class. Hence, the question arises whether the two classes have a meaningful distinction with respect to CQ query containment. In this subsection, we take the TR-Lineage semantics, the Data and CBase containment semantics and show that the last two may be more desirable in certain situations. In addition we show that TR-lineage semantics are not CQ-equivalent with Data (or CBase) semantics.

Continuing our Example 6, let us consider again queries $Q_1(x) : -Drives(x, y), Saw(z, y)$ and $Q_2(x) : -Drives(x, y)$. The result of query Q_2 over U will again be a ULDB $Q_2(U)$ which will include U along with the ULDB relation R_{Q_2} shown in Figure 3.8. In Figures 3.4 and 3.5 we see the result of posing Q_1 and Q_2 over the possible instance D_2 of U. Intuitively Q_1 will return the name of a suspect (a driver that has been seen from a witness) while Q_2 will return the names of all drivers. Sometimes we care only about the fact that if someone is a suspect then he also drives a car. It will then be natural to have that Q_1 is contained in Q_2. Our #1 semantics of Data concern this kind of containment and we have that $Q_1 \subseteq_{Data} Q_2$ holds.

We now consider "contained base lineage" (CBase) containment. It is natural to consider that we have LDB Database containment between two LDBs D_1 and D_2 if for every tuple with data t_1 and base lineage λ_{B1} in D_1 there exists a tuple in D_2 with same data and with a "relaxed" base lineage $\lambda_{B2} \subseteq \lambda_{B1}$. We will show in Section 3.4 that this kind of LDB containment is useful when we have unreliable base data. We will also prove in the same section that even though our first two kinds of Data and CBase containment are not equivalent for LDB database containment, they are equivalent for LDB and ULDB Query containment. So which semantics are more suitable depends on whether it is important in our application to track lineage connections.

Observe that if we adopt the definition of Trio's LDB TR-containment (presented in Section 3.5) we will have that $Q_1 \subseteq_{TR} Q_2$ does not hold. The reason is that Trio LDB containment requires the containment of all lineage (not only base) of the contained relation through the transitive closure of lineage of the containing relation. In our example let us denote with D'_1 the LDB database shown in Figure 3.4 which is relation R_{Q1} of the answer of Q_1 posed on possible instance D_2 and with D'_2 the LDB database shown in Figure 3.5 which is the relation R_{Q2} of Q_2 posed on D_2. In D'_1 there exists a tuple with data *John* and lineage pointing to both alternatives $(11, 2)$ and $(21, 2)$. TR-lineage LDB containment between D'_1 and D'_2 would require that a tuple with data *John* will also exist in D'_2 and that the transitive closure of its lineage in LDB D_2 will contain not only base $(11, 2)$, but non-base $(21, 2)$ as well. Hence TR-lineage LDB containment does not hold for the answers of conjunctive queries Q_1 and Q_2 posed on the second possible instance of U. As a result we do not have TR-lineage CQ query containment between Q_1 and Q_2.

3.3 Basic Definitions on Containment

We study conjunctive query containment and equivalence not only for conjunctive queries, but also for unions of conjunctive queries (UCQs). A *union of conjunctive queries* (UCQ) $\bar{Q}$ is a set $\bar{Q} = \{Q_1, Q_2, \ldots, Q_n\}$, where each Q_i is a conjunctive query. In order to compute a UCQ $\bar{Q} = \{Q_1, Q_2, \ldots, Q_n\}$ over a ULDB U we simply compute $Q_i(U)$ for every $i = 1 \ldots n$ and return in $\bar{Q}(U)$ the ULDB relations of U along with a new relation $R_{\bar{Q}}$.

ULDB relation $R_{\bar{Q}}$ includes the union of tuples with data t and lineage $\lambda(t)$ occurring in R_{Q_i} of every $Q_i(U)$ such that tuples with same data and lineage are merged and x-tuples are then formed using the method of step 2 in Algorithm 2, presented on Chapter 2.

The containment tests for conjunctive query containment under our different kinds of semantics will make use of the following definitions: Let Q, Q' be two conjunctive queries, and let h be a mapping from variables and constants of Q' to variables and constants of Q such that h is the identity for constants. We say that h is a *containment mapping* from Q' to Q if $h(\mathit{head}(Q')) = \mathit{head}(Q)$ and every atom in the body of Q' is mapped to an atom of the body of Q with the same predicate. Duplicate atoms are allowed in the query bodies. A containment mapping from Q' to Q is (subgoal) *onto* if we additionally have that the set of images of all the subgoals of Q' contains *every* subgoal of the body of Q.

Some query equivalence tests will also make use of the notion of *Isomorphic* conjunctive queries. Two conjunctive queries Q_1 and Q_2 are isomorphic, denoted with $Q_1 \simeq Q_1$, if there exists a containment mapping between $h: Q_2 \to Q_1$ that is bijective and its inverse h^{-1} is also a containment mapping from $Q_1 \to Q_2$. Hence isomorphic queries are identical up to renaming of non-distinguished variables and up to a reordering of their subgoals.

LDB and ULDB Database and Query Containment:

We present now the formal definitions for the five different kinds of LDB database containment. In the following three sections we will illustrate their meaning in detail, giving motivating examples that highlight their differences and show in which case each of them may be more suitable.

The first kind of Data LDB Containment will use the notion of S_- which we explain in this paragraph. Suppose there are one or more relations in a ULDB $U = (\bar{R}, S, \lambda^U)$ such that none of their identifiers appear in the lineage of some alternative of U. Let S' be the set of identifiers of all x-tuples and alternatives of those relations. If there is no such relation then $S' = \emptyset$. We denote with S_- the symbols of S that are not in S'. If $S' = \emptyset$ then $S_- = S$. The set S_- includes the identifiers of all x-tuples and alternatives in relations that have lineage pointing to them.

This notion is used for the following reason: recall from Algorithm 1 of Chapter 2 that we attach a new unique identifier in every tuple in relation R_Q of the answer of a CQ Q over an LDB D (the same holds also for ULDB identifiers, see Algorithm 2). As a result when we have a given LDB D and pose to it a query Q_1 the result will be a new LDB which will contain D and a new relation R_{Q1} with new identifiers. On the other hand when we pose another query Q_2 over D the result will be a new LDB which will contain D and the LDB relation R_{Q2} again with new identifiers. But the newly created identifiers in R_{Q1} or R_{Q2} may not be the same. When we want to check containment between the LDB answers of Q_1 and Q_2 we just want to check containment between data and lineage. Clearly identifiers do not alter data. As far as lineage is concerned the newly created identifiers do not appear in any lineage (only the opposite holds: the lineage of new tuples refers to existing identifiers of D). This holds more general, i.e., given an LDB the identifiers of relations which never occur in any lineage do not have any effect to its data or its lineage connections. As far as containment between two LDBs D and D' is concerned, if a tuple with data t and identifier ID occurs in a relation R_i of D that has lineage pointing to it then there must exist a tuple with dame data and identifier in relation R_i of D'. We must have condition $S_- \subseteq S'_-$ to hold as well in order to cover possible external identifiers occurring in the set of symbols of D and D'.

With $ID_{R_i}(t)$ we denote the identifier that a tuple with data t has in relation R_i. The formal definitions for those first two kinds of LDB database containment are the following:

Definition 12 (Data LDB Containment $\subseteq_{Data}$)**.** Let $D = (\bar{R}, S, \lambda^D)$ and $D' = (\bar{R}', S', \lambda^{D'})$ be two LDBs, where $\bar{R}$ and $\bar{R}'$ have the same set of relations. We say that D is *Data* LDB-contained in D' (denoted as $D \subseteq_{Data} D'$), if the following two hold:
1. $S_- \subseteq S'_-$.
2. For every $i = 1, 2, \ldots, n$, either of the following hold: a) if $t \in R_i$ and $ID_{R_i}(t)$ is not contained in S_- then there exists a tuple with data t in R'_i.
b) if $t \in R_i$ and $ID_{R_i}(t)$ is contained in S_- then there exists a tuple with data t in R'_i and $ID_{R_i}(t) = ID_{R'_i}(t)$.

The above definition do not impose lineage restrictions. The other four semantics of database containment will concern lineage restrictions as well, hence they will also include a lineage condition $COND_i$, $i = 2 \ldots 5$. The lineage conditions will concern lineage $\lambda(t)$ of the tuple with data t in R_i and the lineage $\lambda'(t)$ of the tuple with data t in R'_i in clauses (2a) and (2b) of Definition 12. Thus, we will define each of these four semantics by the same definition as above with the only modification that we add $COND_i$ in the end of clauses (2a) and (2b) of Definition 12. Note that for Semantics #1 (Definition 12) $COND_1$ is empty. Specifically we have:

Definition 13 (CBase, TR, SBase, Same LDB Containment $\subseteq_L$)**.** The additional condition is:
• $COND_2 : \quad \lambda'_B(t) \subseteq \lambda_B(t),$

for Contained Base Lineage (CBase-lineage) LDB Containment $\subseteq_{CBase}$.
• $COND_3: \quad \lambda(t) \subseteq \lambda'^{*}(t)$,
for Trio-Transitive Closure-Lineage (TR-lineage) LDB Containment $\subseteq_{TR}$. [2]
• $COND_4: \quad \lambda'_B(t) = \lambda_B(t)$,
for Same Base-Lineage (SBase- lineage) LDB Containment $\subseteq_{SBase}$.
• $COND_5: \quad \lambda'(t) = \lambda(t)$,
for Same-Lineage LDB Containment $\subseteq_{Same}$.

A ULDB represents a set of possible instances that are LDBs. Our definition of when two ULDB databases are contained in each other under one of the above five variants depends on the corresponding definitions of LDB database containment $\subseteq_L$. We have:

Definition 14 (ULDB Database Containment $\subseteq_L$)**.** Let $\subseteq_L$ denote one of the variants $\subseteq_{Data}$, $\subseteq_{CBase}$, $\subseteq_{TR}$, $\subseteq_{SBase}$, $\subseteq_{Same}$ of LDB database containment of Definitions 12 or 13.
Let U and U' be two ULDB's. We say that U is UDLB L-contained in U' (denoted with $\subseteq_L$) if the following hold:
i) for every possible instance D_i of U there exists a possible instance D'_j of U' such that: $D_i \subseteq_L D'_j$ holds according to the corresponding notion of LDB containment definition and
ii) for every possible instance D'_j of U' there exists a possible instance D_i of U such that: $D_i \subseteq_L D'_j$ holds.

The above definition of ULDB Database Containment is natural to define for the above five kinds of ULDB containment semantics that are based on LDB containment between pairs of possible instances:
A ULDB represents a set of possible instances and we do not know which one is the one that "captures the truth". If there exists even one possible instance D_k of a ULDB U that is not LDB contained in any possible instance of a ULDB U' then this D_k might be the "correct" one. Hence in this case we will lose the containment between the "true" possible instances. Similarly suppose that there exists one possible instance D'_k of U' such that there exists no possible instance D_k of U such that $D_k \subseteq_L D'_k$. Then again the "true" possible instance of U' might be D'_K for which there will be no possible instance of U that is contained to it (note that the empty database may not be a possible instance of U). To illustrate why this direction is necessary consider ULDBs U_1 and U_2 shown in Figures 3.9 and 3.10. ULDB U_1 has only one possible instance, let us denote it with D_{11}, with one tuple with data

[2]In [BSH+08a] same tuple identifier was required even for relations to which no other relation point through lineage, but we relax this too restricted definition using the notion of S_{-}.

ID	**Saw**(witness,car)
11	Cathy,Honda

Figure 3.9: ULDB U_1

ID	**Saw**(witness,car)
11	Cathy,Honda || Cathy,Mazda

Figure 3.10: ULDB U_2

$(Cathy, Honda)$ and empty lineage. ULDB U_2 has two possible instances: one identical with the possible instance of U_1, let us denote it with D_{21} and another one, say D_{22}, with only tuple $(Cathy, Mazda)$ (and again empty lineage). Clearly direction (i) of Definition 14 holds. If the possible instance of U_2 is indeed D_{21} then containment between identical D_{11} and D_{21} holds. But if the true possible instance of U_2 is D_{22} then even Data LDB containment will not hold between D_{11} and D_{22}. So intuitively we do not want $U_1 \subseteq_L U_2$ to hold for the ULDB databases in Figures 3.9 and 3.10 under any of the other four containment semantics which subsume Data containment. Since we do not know which possible instance of U_2 is the correct one we must make sure that the containment holds between all possible instances of both U_1 and U_2, i.e., condition (ii) of Definition 14 must also hold.

We now give the following definitions of ULDB query containment and equivalence:

Definition 15 (ULDB Query Containment)**.** Let Q_1 and Q_2 be two arbitrary queries and L one arbitrary semantic of ULDB database containment. Query Q_1 is ULDB L-contained in a query Q_2, denoted with $Q_1 \subseteq_L Q_2$, if for every ULDB U we have that: $Q_1(U) \subseteq_L Q_2(U)$, where $Q_1(U)$ and $Q_2(U)$ are the two ULDBs that are the answers of Q_1 and Q_2 over U.

Definition 16 (ULDB Query Equivalence)**.** Let Q_1 and Q_2 be two arbitrary queries and L one arbitrary semantic of ULDB database containment. Query Q_1 is ULDB L-equivalent with a query Q_2, denoted with $Q_1 \equiv_L Q_2$, if for every ULDB U we have that: $Q_1(U) \subseteq_L Q_2(U)$ and $Q_2(U) \subseteq_L Q_1(U)$, where $Q_1(U)$ and $Q_2(U)$ are the two ULDBs that are the answers of Q_1 and Q_2 over U.

For query containment purposes there exists an one-to-one and onto correspondence between the possible instances of two ULDBs, as the following

Proposition 1 shows. Given a ULDB U and two queries Q_1 and Q_2 we say that for every possible instance of $Q_1(U)$ there exists one *corresponding* possible instance of $Q_2(U)$.

Proposition 1. *Let Q_1 and Q_2 be two arbitrary queries, U an arbitrary ULDB and $\subseteq_L$ an arbitrary kind of LDB database containment . Then the following hold:*

i) If $Q_1 \subseteq_L Q_2$ then there exists an one-to-one and onto correspondence between all the possible instances of $Q_1(U)$ and $Q_2(U)$, i.e., if U has n possible instances $D_1, D_2, \ldots, D_n$ then for each $i = [1 \ldots n]$ we have that $Q_1(D_i) \subseteq_L Q_2(D_i)$ holds according to the $\subseteq_L$ notion of LDB containment definition.
ii) If $\subseteq_L$ is one of the variants $\subseteq_{Data}$, $\subseteq_{CBase}$, $\subseteq_{TR}$, $\subseteq_{SBase}$, $\subseteq_{Same}$, then the opposite of (i) also holds. I.e., if $Q_1(D_i) \subseteq_L Q_2(D_i)$ for each $i = [1 \ldots n]$ then $Q_1 \subseteq_L Q_2$.

Proof. i) Suppose that U is a ULDB with n possible instances: $D_1, D_2, \ldots, D_n$. We will show now that for each $i = [1 \ldots n]$ we will have that: $Q_1(D_i) \subseteq_L Q_2(D_i)$. Let D_i be one arbitrary possible instance of U. Let ULDB U' be the ULDB that has only one LDB possible instance which is LDB D_i. Since $Q_1 \subseteq_L Q_2$ holds we have from Definition 15 that for every ULDB U we will have that $Q_1(U) \subseteq_L Q_2(U)$. So for ULDB U' we will also have that $Q_1(U') \subseteq_L Q_2(U')$ or equivalently that $Q_1(D_i) \subseteq_L Q_2(D_i)$ holds.
ii) If $\subseteq_L$ is one of the variants $\subseteq_{Data}$, $\subseteq_{CBase}$, $\subseteq_{TR}$, $\subseteq_{SBase}$, $\subseteq_{Same}$ we have that if $Q_1(D_i) \subseteq_L Q_2(D_i)$ holds for each $i = [1 \ldots n]$, then both conditions of Definition 14 are satisfied. So from Definition 15 we will have that $Q_1 \subseteq_L Q_2$. □

Notice that the two clauses of Proposition 1 refer to different kinds of ULDB containment. In fact, the first clause refers to general ULDB containment that includes the ULDB containment definitions in Section 3.11 [3], whereas the second clause refers only to the definition of ULDB containment of Definition 14. The second clause is not true for the ULDB containment definitions in Section 3.11.

[3]It is actually useful in proofs in Section 3.11

3.4 Semantics #1 (Data containment - $\subseteq_{Data}$) and Semantics #2 (Contained Base Lineage - $\subseteq_{CBase}$).

It is natural for ULDB query containment between two queries Q and Q' to require that for every ULDB U, we will have that if a tuple with data t exists in relation R_Q of a possible instance of $Q(U)$ then a tuple with same data t will also exist in relation $R_{Q'}$ in the corresponding possible instance of $Q'(U)$. Our first Data containment (Definition 12) is based on this kind of semantics which are useful in situations where we care only about data, as we saw in subsection 3.2.1. The other kinds of database containment retain this natural condition of contained data and additionally have lineage constraints. In a possible LDB instance of a ULDB we allow duplicates of data only if they have different lineage (LDB Definition 1). With LDB database containment under Data semantics where lineage is ignored we may have the case where a tuple with data t appears (with different lineage) in a relation R_i of the contained LDB D more times than the times it appears in the containing LDB D' (again with different lineage). In order for this to happen we must have that the identifiers of tuples in relation R_i of the contained LDB D do not belong in S_{-}. Thus we only want that if a tuple with data t appears in a contained LDB D then containing LDB D' will also have t in it. On the other hand if identifiers of R_i belong in S_{-} only R_i of containing LDB D' can contain more tuples with same data than R_i of D because the identifier of a tuple with data t in D must also be the identifier of a tuple with data t in D'. This subtlety does not affect query containment between two queries Q_1 and Q_2 since both relations R_{Q_1} and R_{Q_2} have no lineage pointing to them, so their identifiers do not belong in S_{-}.

As we saw on Definition 2 an LDB tuple t^{LDB} has a "data part" (a tuple t), a unique identifier and a lineage function that connects it to other LDB tuples through the set of their identifiers. So, depending on the application, we may define LDB containment considering not only data, but lineage as well: From Definition 13 we have CBase LDB database containment between two LDBs D_1 and D_2 if for every tuple with data t_1 and base lineage λ_{B1} in D_1 there exists a tuple in D_2 with same data and with base lineage $\lambda_{B2} \subseteq \lambda_{B1}$. The containment of base lineage goes from the containing database D' to the contained D. This is the opposite way with the containment of data (Definition 12) and with the lineage containment condition of Trio's TR-lineage containment.

CBase LDB containment is important to consider in the practical cases where we can have that a base tuple can be unreliable. The reason is that

ID	**Drives**(name,car)
11	John,Mazda

ID	**Saw**(witness,car)
21	Cathy,Mazda

$\lambda(21) = \{(11)\}$

Figure 3.11: LDB D_1.

ID	**Drives**(name,car)
11	John,Mazda
12	Kate,Mazda

ID	**Saw**(witness,car)
21	Cathy,Mazda

$\lambda'(21) = \{(11), (12)\}$

Figure 3.12: LDB D'_1.

it "preserves" reliability: if a tuple is reliable in D_1 because it does not come from an unreliable source then it will also be reliable in D_2. This is illustrated in the following example:

Example 7. Consider LDB instances D_1 and D'_1, shown in Figures 3.11 and 3.12 respectively. Suppose that in the first case *Cathy* was sure that she saw *John* driving, while in D'_1 she stated that she saw both *John* and *Kate*. Those connections are again modeled through lineage. We trivially have $D_1 \subseteq_{Data} D'_1$. On the other hand $D_1 \not\subseteq_{CBase} D'_1$ because the base lineage of tuple $(Cathy, Mazda)$ in D'_1 contains tuple 12 which is not contained in the base lineage of any tuple with data $(Cathy, Mazda)$ in relation Saw of D_1. If base tuple 12 is considered unreliable then tuple $(Cathy, Mazda)$ would be reliable in D_1 but unreliable in D'_1. So the lineage connection between tuples 21 and base tuple 12 is important and should not be ignored. If, for example, a system deletes unreliable tuples then not even data containment would hold between D_1 and D'_1. As a result this is an example where semantics #1 of LDB database containment are not suitable and justify the need of introducing semantics #2.

Data and CBase-lineage query containment definitions follow from general Definition 15 if we replace $\subseteq_L$ with $\subseteq_{Data}$ and $\subseteq_{CBase}$. Since ULDB query containment follows from Definition 14 of ULDB database containent which reasons on possible instances, it is expected that even in semantics #1 the lineage will affect query containment. This is reasonable if we remember that the possible instances of an uncertain database depend on base lineage. Specifically, given two ULDBs U and U', a tuple with data t in a Possible Instance (PI) D_i will appear in the same corresponding PI D'_i of the containing

U' if $\lambda'_B(t) \subseteq \lambda_B(t)$ (see Theorem 1). It turns out that for ULDB conjunctive query computing purposes the query containment test for the above two notions is the same. This result also holds for LDB query containment, since an LDB can be thought as a ULDB with only one alternative on each x-tuple (and itself as its single possible instance). We have the following Theorem:

Theorem 3. *Suppose that Q_1 and Q_2 are two conjunctive queries. Then the following are equivalent:*
i) $Q_1 \subseteq_{Data} Q_2$.
ii) $Q_1 \subseteq_{CBase} Q_2$.
iii) there exists a containment mapping h: $Q_2 \to Q_1$ (not necessarily onto).

Proof. Let U be an arbitrary ULDB and Q_1, Q_2 two CQs:
(iii$\to$ii): Let μ : $Q_2 \to Q_1$ be a containment mapping. We want to show that $Q_1 \subseteq_{CBase} Q_2$. From Proposition 1 it suffices to show that for any arbitrary PI D_k of U the corresponding PIs $Q_1(D_k)$ and $Q_2(D_k)$ are CBase LDB contained according to Definition 13. Let D_k be an arbitrary PI of U. The first condition $S_- \subseteq S'_-$ of CBase containment holds because relations R_{Q_1} and R_{Q_2} do not have any lineage pointing to them, so their identifiers do not belong in S_- and LDB relations of D_k are the same (so with same identifiers) in both $Q_1(D_k)$ and $Q_2(D_k)$. It now remains to show that if a tuple with data t exists in $Q_1(D_k)$ then a tuple with data t and contained base lineage exists in $Q_2(D_k)$. From Algorithm 2 of computing a CQ over a ULDB we have that an alternative with data t exists in ULDB relation R_{Q1} of $Q_1(U)$ if there exists a mapping σ from the body of Q_1 to the alternatives of U. Its lineage is the union of the identifiers of the alternatives that are the images of Q_1 under this mapping. Then $\sigma \circ \mu$ is a substitution for variables of Q_2. Because μ is a containment mapping $\sigma \circ \mu$ maps the subgoals of Q_2 to a subset of the subgoals of Q_1. According to Algorithm 2, because we have a containment mapping between Q_2 and Q_1, the substitution $\sigma \circ \mu$ will add to ULDB relation R_{Q2} of $Q_2(U)$ an alternative with data t and lineage $\lambda_2(t) \subseteq \lambda_1(t)$. Expanding lineages λ_1 and λ_2 back to base data we have: $\lambda_{B2}(t) \subseteq \lambda_{B1}(t)$. We have that D_k is an arbitrary PI of U. So D_k occurs due to the selection of a set λ_B of base tuples. If an alternative with data t and and lineage $\lambda_1(t)$ occurs in the answer of $Q_1(D_k)$ then it will also occur in the answer of $Q_2(D_k)$, according to Theorem 1, because: i) PI D_k will have selected the same base alternatives, ii) $\lambda_{B2}(t) \subseteq \lambda_{B1}(t)$ and iii) answers of CQs have always well-behaved lineage pointing to existing tuples.

(i$\to$iii): Suppose that $Q_1(U) \subseteq_{Data} Q_2(U)$ holds for every ULDB U. Let D_k be an arbitrary PI of U. From clause (i) of Proposition 1 we have that: $Q_1(D_k) \subseteq_{Data} Q_2(D_k)$. Since $Q_1(U) \subseteq_{Data} Q_2(U)$ holds for every ULDB U, it must hold also for any ULDB that has empty lineage. Any possible instance of such a ULDB is an ordinary database with no lineage and no data duplicates. As a result Q_1 must be query contained in Q_2 for ordinary databases. In [CM77] it was shown that this is true only if there exists a containment mapping from conjunctive query Q_2 to Q_1.

(ii→i): Suppose that $Q_1(U) \subseteq_{CBase} Q_2(U)$ holds. Let D_k be an arbitrary PI of U. From Proposition 1 clause (i) we have that: $Q_1(D_k) \subseteq_{CBase} Q_2(D_k)$. Since CBase containment includes the data constraint of Data containment we have that $Q_1(D_k) \subseteq_{Data} Q_2(D_k)$ also holds. From Proposition 1 clause (ii), we have that: $Q_1 \subseteq_{Data} Q_2$, because U and D_k are arbitrary. □

Note that, in contrast with conjunctive query containment, for LDB database containment the first two semantics are not equivalent (see Example 7). For the ULDB U and CQs of Example 6 we have that there exists a containment mapping from Q_2 to Q_1 and indeed ULDB $Q_1(U)$ is both Data and CBase contained in ULDB $Q_2(U)$ as the following example shows. Since there exists a containment mapping between the two queries CBase and Data Query containment will hold for every ULDB and not only for the one in our example.

Example 8. For the ULDB U and CQs of Example 6 we have that Q_1 is CBase-lineage and Data query contained in Q_2 as we intuitively expected. This holds because the mapping which maps variables x and y to themselves is a containment mapping from Q_2 to Q_1. Indeed for every possible instance D_i of this ULDB U we have $Q_1(D_i)$ is CBase and Data LDB contained in $Q_2(D_i)$. For the second possible instance D_2 (Figure 3.3) of U we have: The instances of relations R_{Q1} and R_{Q2} (shown in Figures 3.4, 3.5), referring to the possible instance D_2 both contain a tuple with data *John*. So $Q_1(D_2) \subseteq_{Data} Q_2(D_2)$ holds. For tuple $(31, 2)$ with data *John* that has base lineage equal with $(11, 2)$ in $Q_1(D_2)$ (note that alternative $(21, 2)$ is not base and points again to base $(11, 2)$) there exists tuple $(41, 2)$ in $Q_2(D_2)$ with data *John* and base lineage contained in the base lineage of $(31, 2)$. Actually in this example base lineage is not only contained but equal, i.e., we have that $\lambda_B(41, 2) = \{(11, 2)\} = \lambda_B(31, 2)$. Similarly for tuple $(32, 1)$ with data *John* and $\lambda_B(32, 1) = \{11, 2\}$ there exists in $Q_2(D_2)$ tuple again $(41, 2)$ with same data and contained (in our example again equal) base lineage. In addition identifiers in relations R_{Q1} and R_{Q2} do not belong to S_-, while relations of U whose identifiers can be in S_- are the same in $Q_1(U)$ and $Q_2(U)$. Hence $Q_1(D_2) \subseteq_{CBase} Q_2(D_2)$ holds. It is easy to check that LDB CBase containment also holds for the answers shown in Figures 3.13, 3.14 of the two queries posed on the first possible instance D_1 of U. As a result from Definition 14 of ULDB Database Containment we have that $Q_1(U) \subseteq_{CBase} Q_2(U)$.

ID	R_{Q_1}
31,1	John
33,1	Kate

$\lambda(31, 1) = \{(11, 1), (21, 1)\}$
$\lambda(33, 1) = \{(12, 1), (21, 1)\}$

Figure 3.13: D_{Q11}: LDB relation R_{Q_1} for: $Q_1(x) : -Drives(x, y), Saw(z, y)$ posed on D_1.

ID	R_{Q_2}
41,1	John
42,1	Kate
43,1	Kate

$\lambda(41, 1) = \{(11, 1)\}$
$\lambda(42, 1) = \{(12, 1)\}$
$\lambda(43, 1) = \{(13, 1)\}$

Figure 3.14: D_{Q21}: LDB relation R_{Q_2} for: $Q_2(x) : -Drives(x, y)$ posed on D_1.

In [CM77] it was shown that checking for the existence of a containment mapping between two conjunctive queries is NP-complete. Hence we have:

Corollary 1. *Suppose that Q_1 and Q_2 are two conjunctive queries. Then the following holds: Checking whether $Q_1 \subseteq_{Data} Q_2$ or $Q_1 \subseteq_{CBase} Q_2$ is NP-complete.*

3.5 Semantics #3: Trio/Transitive Closure of Lineage Containment (TR-lineage - $\subseteq_{TR}$).

TR-lineage LDB containment was firstly defined in Trio [BSH+08a]. For two LDBs D_1 and D_2 it requires that data in D_1 is contained in the data of D_2 and all the lineage connections of LDB D_1 are preserved in the transitive closure of the lineage of D_2. With λ^* we denote the transitive closure of lineage λ. Specifically the lineage constraint for TR-lineage LDB database containment between two LDBs D_1 and D_2 is that for every tuple with data t_1 and lineage $\lambda_1(t)$ in D_1 there exists a tuple in D_2 with same data and with lineage that contains $\lambda_1(t)$ in its transitive closure, i.e. $\lambda_1(t) \subseteq \lambda_2^*(t)$.

ID	**Drives**(name,car)
11	John,Mazda

ID	**Saw**(witness,car)
21	Cathy,Mazda

$\lambda(21) = \{(11)\}$

ID	**Color**(car,color)
31	Mazda, Blue

$\lambda(31) = \{(21)\}$

Figure 3.15: LDB D_2.

ID	**Drives**(name,car)
11	John,Mazda

ID	**Saw**(witness,car)
21	Cathy,Mazda

$\lambda'(21) = \{(11)\}$

ID	**Color**(car,color)
31	Mazda, Blue

$\lambda'(31) = \{(11)\}$

Figure 3.16: LDB D'_2.

TR-lineage LDB containment is also important to consider in the practical cases where we can have that a non-base tuple can be unreliable. The reason is that it "preserves" unreliability: if a tuple is unreliable in D_1 then it will also be unreliable in D_2. In such cases we need to "retain" all lineage of the contained database through the transitive closure of lineage of the containing database:

Example 9. Consider LDB instances D_2 and D'_2, shown in Figures 3.15 and 3.16 respectively. Trivially we have $D_2 \subseteq_{Data} D'_2$ and we can check that $D_2 \subseteq_{CBase} D'_2$ also holds. On the other hand $D_2 \not\subseteq_{TR} D'_2$ because the lineage connection between tuples $(Mazda, Blue)$ and $(Cathy, Mazda)$ in D_2 is not retained in D'_2. If non-base tuple 21 is considered unreliable (but other tuples in the database before posing the queries are considered reliable) then tuple $(Mazda, Blue)$ would be unreliable in D_2 but reliable in D'_2. So the non-base lineage connection between tuples 21 and 31 is important and should not be ignored. As a result this is an example where semantics #1 and #2 of LDB database containment are not suitable and justify the need of introducing semantics #3.

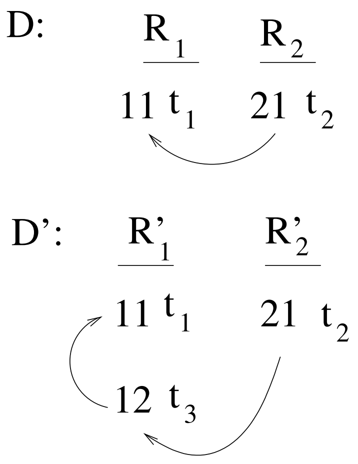


Figure 3.17: Example of TR-lineage LDB containment $D \subseteq_{TR} D'$.

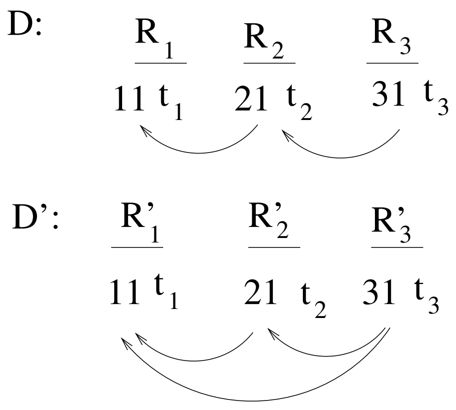


Figure 3.18: Example of $D \subseteq_{TR,SBase} D'$ and $D' \subseteq_{TR,SBase} D$.

In Figure 3.17 we see an example of TR-lineage LDB containment (for clarity, lineage is represented through arrows). It is easy to check that $D \subseteq_{TR} D'$. In Figure 3.18 we see an example of two LDBs D and D' for which it holds that $D \subseteq_{TR} D'$ and $D' \subseteq_{TR} D$. For conjunctive query TR-containment we have the following Theorem which follows from Theorem 5 proven in the next section:

Theorem 4. *Given two conjunctive queries Q_1 and Q_2 we have that $Q_1 \subseteq_{TR} Q_2$ iff there exists a subgoal-onto containment mapping μ: $Q_2 \rightarrow Q_1$.*

In [CV93] it was shown that checking for a subgoal-onto containment mapping between two conjunctive queries is NP-complete. Hence from the above theorem we have:

Corollary 2. *Suppose that Q_1 and Q_2 are two conjunctive queries. Then the following holds: Checking whether $Q_1 \subseteq_{TR} Q_2$ is NP-complete.*

We now present an example of two onto-contained conjunctive queries and show that for the ULDB U and CQs of Example 6 we have TR-lineage query containment (as we will have for every ULDB):

Example 10. We continue Example 6. Consider another query $Q_3(x)$:-$Drives(x, y), Saw(z, y), Saw(w, y)$. The answer of Q_3 posed in ULDB U will return the names of persons who drive a car that was shown near the crime-scene and with lineage pointing at one or two witnesses. The result of this query

ID	R_{Q_3}
51	John || John
52	John '?'
53	Kate '?'
54	John '?'

$$\lambda(51,1) = \{(11,1),(21,1)\}$$
$$\lambda(51,2) = \{(11,2),(21,2)\}$$
$$\lambda(52,1) = \{(11,2),(22,1)\}$$
$$\lambda(53,1) = \{(12,1),(21,1)\}$$
$$\lambda(54,1) = \{(11,2),(21,2),(22,1)\}$$

Figure 3.19: ULDB Relation R_{Q_3} for $Q_3(x) : -Drives(x,y), Saw(z,y)$, $Saw(w,y)$.

ID	R_{Q_3}
51,1	John
53,1	Kate

$$\lambda(51,1) = \{(11,1),(21,1)\}$$
$$\lambda(53,1) = \{(12,1),(21,1)\}$$

Figure 3.20: D_{Q31}: First Possible Instance of ULDB relation R_{Q_3}.

ID	R_{Q_3}
51,2	John
52,1	John
54,1	John

$$\lambda(51,2) = \{(11,2),(21,2)\}$$
$$\lambda(52,1) = \{(11,2),(22,1)\}$$
$$\lambda(54,1) = \{(11,2),(21,2),(22,1)\}$$

Figure 3.21: D_{Q32}: Second Possible Instance of ULDB relation R_{Q_3}.

over U (according to Algorithm 2) will be a ULDB U_3 which will include ULDB relations `Saw` and `Drives` of U along with the ULDB relation R_{Q_3} shown in Figure 3.19. The possible instances of relation R_{Q_3} are the LDBs D_{Q31} and D_{Q32}, shown in Figures 3.20 and 3.21. The first possible instance of $U_3 = Q_3(U)$ includes LDB relations of D_1 (first possible instance of U, shown in Figure 3.2) together with LDB relation D_{Q31}. Accordingly the second PI of U_3 consists of D_2 (shown in Figure 3.3) and D_{Q32}. Note that for ordinary databases without lineage or uncertainty we have both that $Q_1 \subseteq Q_3$ and $Q_3 \subseteq Q_1$, but when we take lineage into account as well only the former containment holds: Remember that Q_1 asks for suspects that were shown from a witness and so every alternative in R_{Q1} has lineage pointing at one witness. Hence we expect $Q_1(U)$ to be ULDB contained to $Q_3(U)$ since a relation with an alternative with lineage pointing to one witness is expected to be contained to a relation with an alternative with lineage pointing to one witness and an alternative with lineage pointing to two witnesses. Indeed for the first possible instance of query Q_1 posed over D_1, shown in Figure 3.13 we have that there exists a tuple with data $John$ and lineage equal with the set $\{(11,1),(21,1)\}$. Tuple $(51,1)$ in the possible instance of Q_3 shown in Figure 3.20 has equal data and lineage. The same holds also for the second tuple $(33,1)$ of Figure 3.13 and tuple $(53,1)$ of Figure 3.20. Hence TR-lineage holds for the first possible instance of $Q_1(U)$ and $Q_3(U)$. It is easy to check that this is true also for the second possible instance of $Q_1(U)$, shown in Figure 3.4, and the second possible instance of $Q_3(U)$, shown in Figure 3.21. TR-lineage containment between Q_1 and Q_3 holds for every ULDB since it is easy to check that the mapping $x \to x$, $y \to y$, $z \to z$, $w \to z$ is an onto containment mapping from Q_3 to Q_1.

3.6 Semantics #4 (Same Base Lineage - $\subseteq_{SBase}$) and Semantics #5 (Same Lineage Containment - $\subseteq_{Same}$).

Consider LDBs D_3 and D_3' of Figures 3.22, 3.23. The base lineage of a tuple includes all the base tuples that appear in the transitive closure of its lineage (see Definition 5). With λ^{D_3} we denote the lineage function of D_3 and with $\lambda^{D_3'}$ the lineage of D_3'. We have $D_3 \not\subseteq_{TR} D_3'$ because the lineage connection between tuples 31 and 21 ($\lambda(31) = \{21\}$) in D_3 is not retained in the transitive closure $\lambda'^*(31)$ of D_3'. On the other hand they have the same base lineage: $\lambda_B(31) = \lambda_B'(31) = \{11, 12\}$. Suppose that we consider reliable $HighSuspects$ only the ones that were seen from two different witnesses, i.e.,

ID	**Saw**(witness,car)
11	Cathy,Mazda
12	Amy,Mazda

ID	**Suspect**(name)
21	John

$\lambda(21) = \{11, 12\}$

ID	**High-Suspect**(name)
31	John

$\lambda(31) = \{21\}$

Figure 3.22: LDB D_3.

ID	**Saw**(witness,car)
11	Cathy,Mazda
12	Amy,Mazda

ID	**Suspect**(name)
21	John

$\lambda'(21) = \{11, 12\}$

ID	**High-Suspect**(name)
31	John

$\lambda'(31) = \{11, 12\}$

Figure 3.23: LDB D'_3.

come from two different base tuples. In this case both tuples 31 in D_3 and D'_3 would be reliable. Then the information about the connection between tuples 31 and 21 (which is non-base) will not be important and we intuitively should have LDB database containment between D_3 and D'_3. On the other hand the fact that both 31 tuples have 11 and 12 as base lineage should not be ignored.

In situations like this it is natural to require for the data of an LDB D to be included in the data of an LDB D' with exactly the same base lineage. Such a condition is also important in ULDB data exchange purposes, for containment of certain answers [AV10a]. The SBase-lineage semantics (Definition 13) capture this requirment by making sure that our tuples in the containing relation come from the same set of base tuples. Note that CBase containment semantics do not suffice for this case:

Example 11. Consider again LDB D_3 of Figure 3.22 and now an LDB D''_3 with same data and IDs but this time with $\lambda''(21) = \{11\}$ (and again $\lambda''(31) = \{11\}$). In this case tuple 31 in D''_3 should not be considered reliable because it comes only from base tuple 11. But we have $D_3 \subseteq_{CBase} D''_3$ because $\lambda''_B(31) = \{11\} \subseteq \{11, 12\} = \lambda_B(31)$. On the other hand we correctly do not have SBase containment: $D_3 \not\subseteq_{SBase} D''_3$.

We also note that TR-lineage containment does not imply SBase containment, as the following example shows:

Example 12. For LDBs D_1 and D'_1 shown on Figures 3.11 and 3.12 we have TR-lineage containment but not SBase containment. TR-lineage containment holds because the lineage connection $\lambda(21) = \{11\}$ is preserved in λ'. On the other hand $\lambda_B(21) = \{11\} \neq \{11, 12\} = \lambda'_B(21)$.

The more strict kind of containment (Same Lineage - Definition) is requiring data containment with exactly the same lineage (not only base). It is important in cases where we want to combine the conditions of TR-lineage and SBase lineage. So when we want both to: i) retain all the lineage and not only the base one (like in TR-lineage containment), and ii) we do not want to have lineage connections to a tuple from a base tuple that exists only in the containing relation (we did not want this to happen in SBase-lineage - see Example 12). It is easy to check that SBase lineage containment does not imply Same lineage containment:

Example 13. For the LDBs D_2 and D'_2, shown in Figures 3.15 and 3.16 we have SBase-lineage containment $D_2 \subseteq_{SBase} D'_2$ (because $\lambda_B(31) = \{11\} = \lambda'_B(31)$) but not Same-lineage containment ($D_2 \not\subseteq_{Same} D'_2$). The reason is that $\lambda(31) = \{21\}$ is not equal with $\lambda'(31) = \{11\}$. Note also that because 21 is not in $\lambda'^*(31)$ TR-lineage LDB containment does not hold ($D_2 \not\subseteq_{TR} D'_2$).

It turns out that SBase-lineage conjunctive query containment holds between two CQs Q_1 and Q_2 if and only if there exists an onto containment mapping from Q_2 to Q_1, like in TR-lineage CQ containment. The same holds also for Same-lineage CQ containment, as the following Theorem 5 shows. It will make use of the following Lemma which states that Same ULDB (and LDB) database containment implies both TR and SBase database containment. SBase-lineage containment for LDBs is equivalent with the *Why*-semiring containment that is defined in [Gre09] for databases with lineage, where it was shown that Why-semiring CQ containment between two conjunctive queries Q_1 and Q_2 holds if and only if there exists an onto containment from Q_2 to Q_1.

Lemma 1. *For all ULDBs U_1 and U_2 we have that the following holds:*
$U_1 \subseteq_{Same} U_2$ implies that $U_1 \subseteq_{TR} U_2$ and that
$U_1 \subseteq_{SBase} U_2$.

Proof. Let U_1 and U_2 be two arbitrary ULDBs. Suppose that $U_1 \subseteq_{Same} U_2$ holds. Let D_{k1} be a possible instance of U_1. From ULDB containment Definition 14 we have that there exists a possible instance of U_2, let us denote it with D_{k2}, such that $D_{k1} \subseteq_{Same} D_{k2}$ holds. Let a tuple with data t and lineage $\lambda_1(t)$ belong to D_{k1}. From Same containment we have that there exists a tuple in D_{k2} with same data t and lineage $\lambda_2(t) = \lambda_1(t)$. Similarly if D_{k2} is a possible instance of U_2 there exists a possible instance D_{k1} of U_1 which has a tuple with same data and lineage.

$(U_1 \subseteq_{Same} U_2) \Rightarrow (U_1 \subseteq_{TR} U_2)$: Let $ID_1 \in \lambda_1(t)$. We want to show that $ID_1 \in \lambda_2^*(t)$. Because $ID_1 \in \lambda_1(t)$ we have that ID_1 belongs to a relation that has lineage pointing to it, so to the set S_- of D_{k1}. From the fact that $\lambda_2(t) = \lambda_1(t)$ we have that ID_1 belongs to $\lambda_2(t)$ (and to S_- of D_{k2}). So $ID_1 \in \lambda_2(t)$ and hence $ID_1 \in \lambda_2^*(t)$.

$(U_1 \subseteq_{Same} U_2) \Rightarrow (U_1 \subseteq_{SBase} U_2)$: Let $ID_1 \in \lambda_1(t)$. Similarly with the above case, we have that ID_1 belongs to the set S_- of D_{k1}, to $\lambda_2(t)$ and to S_- of D_{k2}. If ID_1 is a base tuple identifier then it will also belong to the base lineage of $\lambda_2(t)$. For the same reason if ID_2 is a base tuple identifier in $\lambda_2(t)$ it will also belong to the base lineage of $\lambda_1(t)$. So in this case $U_1 \subseteq_{SBase} U_2$ holds. Otherwise let a base tuple with identifier ID_3 and data t' occur if we expand $\lambda_1(t)$ one level. Then from Same-containment we will have that a base tuple with same identifier ID_3 also exists if we expand $\lambda_2(t)$ one step. If k is the distance of t to the base data of D_{k1} through $\lambda_1^*(t)$, then we can repeat this procedure k times and yield that $D_{k1} \subseteq_{SBase} D_{k2}$. Similarly the same will hold for each base tuple occurring in the base lineage of $\lambda_2(t)$.

□

Theorem 5. *Given two conjunctive queries Q_1 and Q_2 we have that the following are equivalent:*
i) there exists an onto containment mapping
μ: $Q_2 \rightarrow Q_1$.
ii) $Q_1 \subseteq_{TR} Q_2$.
iii) $Q_1 \subseteq_{SBase} Q_2$.
iv) $Q_1 \subseteq_{Same} Q_2$.
The complexity of all the above CQ query containment tests is NP-complete.

Proof. Let U be an arbitrary ULDB and Q_1, Q_2 two CQs. Let D_k be an LDB possible instance of U. Let D_{k1} be the possible instance of $Q_1(U)$ that contains the answer of Q_1 posed on D_k. Let D_{k2} be the possible instance of $Q_2(U)$ that contains the answer of Q_2 posed on the same D_k. Let L be one of TR, SBase, or Same LDB-containmnent semantics. From Proposition 1 we have that $Q_1 \subseteq_L Q_2$ holds if and only if $Q_1(D_k) \subseteq_L Q_2(D_k)$ or, equivalently, $D_{k1} \subseteq_L D_{k2}$ holds.
($i \rightarrow ii$): Let μ : $Q_2 \rightarrow Q_1$ be an onto containment mapping. We want to show that $D_{k1} \subseteq_{TR} D_{k2}$. Condition $S_- \subseteq S'_-$ of TR containment holds because relations R_{Q1} and R_{Q2} of D_{k1} and D_{k2} have no lineage pointing to their tuples. Hence their identifiers do not belong to S_- and do not affect containment. In addition all tuples in relations of D_k have the same identifiers in both $Q_1(D_k)$ and $Q_2(D_k)$. Now in order to have $D_{k1} \subseteq_{TR} D_{k2}$ we must have from Definition 13 of LDB TR-containment that for each tuple with data t in D_{k1} with lineage $\lambda_1(t)$ there exists in D_{k2} a tuple with same data t and with lineage $\lambda_2(t)$ such that: $\lambda_1(t) \subseteq \lambda_2^*(t)$. Without loss of generality let Q_1 have n_1 subgoals and Q_2 $n_2 \geq n_1$ subgoals. From Theorem 3 because μ is a containment mapping we will have that $D_{k2} \subseteq_{Data} D_{k2}$. So if a tuple with data t exists in D_{k1} then a tuple with data t will also exist in D_{k2}. For the lineage we have: A tuple with data t and lineage $\lambda_1(t)$=$\{ID_1, ID_2, \ldots, ID_{n_1}\}$ in LDB relation R_{Q1} of D_{k1} is produced by some substitution σ on the variables of Q_1 that maps all n_1 subgoals of Q_1 to tuples of D_k, one with ID_1, one with ID_2, etc until one with ID_{n_1}. Then $\sigma \circ \mu$ is a substitution for variables of Q_2. Because μ is onto $\sigma \circ \mu$ maps the subgoals of Q_2 to the same set of tuples of D_k with the same set of identifiers $\{ID_1, \ldots, ID_n\}$. So, according to Algorithm 1 on Chapter 2, because we have a subgoal onto containment between Q_2 and Q_1, the substitution $\sigma \circ \mu$ will add to LDB relation R_{Q2} of D_{k2} a tuple with data t and lineage $\lambda_2(t) = \lambda_1(t)$. Since we have equality and lineages $\lambda_2(t)$ and $\lambda_1(t)$ points to LDB relations in D_k, $\lambda_1(t) \subseteq \lambda_2^*(t)$ will also hold as well.

($iii \rightarrow i$): Suppose that $D_{k1} \subseteq_{SBase} D_{k2}$ holds. From Definition 13 for LDB SBase-containment we have that if there exists a tuple with data t in relation R_{Q1} of D_{k1} with base lineage $\lambda_{B1}(t)$ then there exists in relation R_{Q2} of D_{k2} a tuple with same data t and with base lineage $\lambda_{B2}(t)$ such that: $\lambda_{B1}(t) \subseteq \lambda_{B2}(t)$. In order to have the containment only for data from Theorem 3 we must have that there exists a (not necessarily onto) containment mapping from Q_2 to Q_1.

We will now prove the contrapositive: we will suppose that there exists no onto containment mapping and prove that in this case SBase-containment does not hold. Suppose that there exists no onto containment mapping from Q_2 to Q_1. We will show that in this case $D_{k1} \not\subseteq_{SBase} D_{k2}$. Let h be a containment mapping from Q_2 to Q_1. From our assumption h will not be onto. From Definition of onto containment mapping we will have that for every containment mapping from Q_2 to Q_1 there will exist some subgoals of Q_1 that are not in the set of images of all the subgoals of Q_2. So let $R_{j1}, R_{j2}, \ldots, R_{jl}$ be the predicates of the subgoals of Q_1 that are not in the set of the images of the subgoals of Q_2 under h. Let $ID_{j1}, ID_{j2}, \ldots, ID_{jl}$ be the set of the identifiers of tuples of D_k that are the images under h of $R_{j1}, R_{j2}, \ldots, R_{jl}$ subgoals of Q_1. The important observation for our proof is the following: Since SBase query containment must hold for *every* ULDB, we can have that all tuples of D_k that are the images under h of $R_{j1}, R_{j2}, \ldots, R_{jl}$ subgoals of Q_1 are *base* tuples. In this case D_{k1} will contain a tuple with data t and base lineage $\lambda_{B1}(t) = \{ID_{j1}, ID_{j2}, \ldots, ID_{jl}\}$. From Algorithm 1 and from the fact that there exists no onto containment mapping from Q_2 to Q_1 there exists no tuple in the answer of Q_2 in D_{k2} with data t and lineage that contains $ID_{j1}, ID_{j2}, \ldots, ID_{jl}$. Since $ID_{j1}, ID_{j2}, \ldots, ID_{jl}$ refer to base tuples, there exists no tuple in the answer of Q_2 in D_{k2} with data t and base lineage equal with $ID_{j1}, ID_{j2}, \ldots, ID_{jl}$.

$(ii \to i)$: Suppose that $D_{k1} \subseteq_{TR} D_{k2}$. From Definition 13 for LDB TR-containment we have that if there exists a tuple with data t in relation R_{Q1} of D_{k1} with lineage $\lambda_1(t)$ then there exists in R_{Q2} of D_{k2} a tuple with same data t and with lineage $\lambda_2(t)$ such that: $\lambda_1(t) \subseteq \lambda_2^*(t)$. In order to have the containment only for data from Theorem 3 we must have that there exists a (not necessarily onto) containment mapping from Q_2 to Q_1.

Similarly with direction $(iii \to i)$ we can prove the contrapositive: if we suppose that there exists no onto containment mapping we can show that in this case TR-containment does not hold: If $ID_{j1}, ID_{j2}, \ldots, ID_{jl}$ are the identifiers of the images of the predicates of the subgoals of Q_1 that are not in the set of the images of the subgoals of Q_2 under a non onto h, then there exists no tuple in the answer of Q_2 in D_{k2} with data t and lineage containing $ID_{j1}, ID_{j2}, \ldots, ID_{jl}$. So TR-containment will not hold. Note that the only way for TR-containment to hold would be to have $ID_{j1}, ID_{j2}, \ldots, ID_{jl}$ contained in the transitive closure of the lineage of the images of the subgoals of Q_2. But since ULDB U is arbitrary we may not assume any lineage connection between its alternatives (e.g., we can have that all alternatives in U have empty lineage). This is illustrated in the following example:

Example 14. Let $Q(x) : -R(x,x)$, $R(x,y)$ and $Q'(x) : -R(x,x)$ be two conjunctive queries. We have that there exists a containment mapping h from Q' to Q but no onto containment mapping exists. Let ULDB U' be the following: It contains a ULDB relation $R(a,b)$ with two x-tuples with IDs 11 and 12. Let x-tuple 11 contain only one alternative with data $(3,3)$ and 12 one alternative with data $(3,2)$. Also suppose that all alternatives have empty lineage and no "?" symbol. This ULDB contains only one possible instance D_k which contains one LDB relation $R(a,b)$ with two tuples which are the only one possible selection from each x-tuple of U. Then $Q(D_k)$ will contain D_k along with relation R_Q which will have two tuples with data 3, the first with lineage $\{11\}$ (from the mapping $x \to 3$ and $y \to 3$) and the second with lineage $\{11,12\}$ (from the mapping $x \to 3$ and $y \to 2$) . On the other hand D'_k will contain D_k along with relation $R_{Q'}$ which will have one tuple with data 3 but lineage $\lambda' = \{11\}$. So we have that $D_k \not\subseteq_{TR} D'_k$ because there exist no tuple in $R_{Q'}$ with data 3 and $\{11,12\}$ in its transitive closure of lineage. This is due to the fact that the second subgoal of Q is not an image of Q' under any containment mapping h. The only way for TR-containment to hold would be to have that 12 is in the transitive closure of $\lambda(11)$, i.e., if we had $\lambda(11) = \{12\}$ in D_k. But since we must have TR-containment for every ULDB U, we may not assume any lineage connection.

$(i \to iv)$: It follows from the proof of direction $(i \to ii)$ where we showed that if there exists an onto containment mapping, then for each tuple with data t in D_{k1} with lineage $\lambda_1(t)$ there exists in D_{k2} a tuple with same data and lineage.

The remaining cases of $(iv \to ii)$ and $(iv \to iii)$ are a consequence of the previous Lemma. The NP-completeness follows from the completeness of checking for a (subgoal) onto containment mapping between two CQs [CV93].

□

3.6.1 Conjunctive Queries without self-joins

A query has self-joins if its body contains at least two subgoals with the same relation name. In [IR95] it was shown that testing for the existence of a (subgoal) onto containment mapping between two conjunctive queries has complexity $O(n \cdot \log n)$. Thus for the case of conjunctive queries without self-joins, the query containment test has $O(n \cdot \log n)$ complexity for all different five semantics.

3.7 Comparison between Different Semantics

In Figure 3.6 of the Introduction we have a comparison of the five notions of containment that we defined. Column *Implies DB cont.* shows implications between different kinds of database containment. For example if a ULDB or LDB database is contained in another under Semantics #4, then it will be contained under Semantics #1 and #2 as well. These implications between different database containment semantics are easy to check from their definitions. For negative implication results we give counter-examples:

In LDBs D_1 and D'_1 of Figures 3.11 and 3.12 we have an example of Data containment and TR-lineage containment between databases (LDBs) D and D', but with non CBase database containment, due to the fact that $\lambda'_B(21) = \{11, 12\} \not\subseteq \{11\} = \lambda_B(21)$ (note that for CBase the base lineage containment goes from the containing to the contained database). For the same reason we do not have same base lineage nor same lineage between D (where $\lambda_B(21) = \{11\}$) and D' (where $\lambda'_B(21) = \{11, 12\}$). In LDBs D_3 and D'_3 of Figures 3.22 and 3.23 we see an example of Data, CBase-lineage and SBase-lineage LDB database containment (because $\lambda_B(31) = \lambda'_B(31) = \{11, 12\}$), but with non TR-lineage nor Same-lineage containment, due to the fact that non-base lineage connection $\lambda(31) = \{21\}$ of D_3 is not retained in D'_3. Lastly, in Example 11 we have an example of CBase-lineage database LDB containment, because the base lineage of 21 in D''_3 is $\{11\}$ which is a subset of the base lineage of 21 in D_3. But in the same Example we do not have SBase-lineage database containment between D_3 and D''_3 due to the fact that the connection $12 \in \lambda_B(21)$ of D_3 is not retained in D''_3. Note that these positive and negative results of database containment implications hold for both LDBs and ULDBs since an LDB can be thought as a ULDB with only one possible instance.

3.8 CQ Query Equivalence

In this section we investigate the problem of ULDB conjunctive query equivalence. From Definition 16 which defines query equivalence as a two-way query containment, Theorems 3 and Corollary 1 the next theorem follows:

Theorem 6. *Given two conjunctive queries Q_1 and Q_2 we have that the following are equivalent:*
i) there exists a containment mapping
h: $Q_2 \to Q_1$ and a containment mapping
h′: $Q_1 \to Q_2$.
ii) $Q_1 \equiv_{Base} Q_2$.

iii) $Q_1 \equiv_{CBase} Q_2$.
Moreover the CQ equivalent test for Base and CBase semantics is NP-complete.

Note that having a ULDB U and two Base or CBase equivalent queries Q and Q' does not necessarily mean that $Q(U)$ and $Q'(U)$ will be two ULDBs with possible LDB instances that will have equal data and lineage. This is illustrated in the following example: Consider queries $Q_1(x)$:- $Drives(x, y), Saw(z, y)$ and $Q_3(x)$:- $Drives(x, y), Saw(z, y), Saw(w, y)$ of Example 10. We have that there exists a containment mapping from Q_1 to Q_3 and from Q_3 to Q_1. So Q_1 and Q_3 are Data and CBase Equivalent. If we look at the second possible LDB instance D_{Q12} of $Q_1(U)$ in Figure 3.4 and at the corresponding second possible LDB instance D_{Q32} of $Q_3(U)$ in Figure 3.21 we see that D_{Q12} contains two different tuples with data *John* while D_{Q32} contains three different tuples with data *John*. But the two ULDBs $Q_1(U)$ and $Q_3(U)$ instances are "equivalent" according to the notion of Data containment: if a tuple with data t occurs in a possible instance of $Q_1(U)$ then a tuple with data t will also exist in the corresponding possible instance of $Q_3(U)$ and the opposite. So Data CQ equivalence means that the sets of tuples between corresponding possible LDB instances of $Q_1(U)$ and $Q_3(U)$ are equal and is a suitable notion of equivalence when we care only about data. Similarly if we replace query Q_3 with its CBase equivalent Q_1 which has less subgoals then we will get in the answer the same set of data tuples that have "minimal" base lineage. That is, if a tuple with data t and base lineage λ_B occurs in a possible instance of $Q_3(U)$ then a tuple with data t and with "minimal" base lineage $\lambda'_B \subseteq \lambda_B$ will exist in the corresponding possible instance of $Q_1(U)$. In cases where CBase containment is useful we care only about containment of base lineage and considering CBase equivalency is meaningful even though two CBase equivalent databases will not have equal data and lineage. As a result for Data or CBase query equivalency the traditional CQ optimization algorithm can be used.

If for two conjunctive queries Q_1 and Q_2 there exists an onto containment mapping $h: Q_2 \to Q_1$ and an onto containment mapping $h': Q_1 \to Q_2$, then Q_1 and Q_2 are isomorphic: Let $\bar{X}_1$ and $\bar{X}_2$ be the sets of all variables that occur in Q_1 and Q_2 respectively. Since there exists an onto containment mapping $h: Q_2 \to Q_1$ we have the head and some subgoals of Q_2 are mapped to the head and all subgoals of Q_1. As a result for all variables of Q_1 there exists a variable of Q_2 that is mapped under h to it, i.e.: $\forall x_{1i} \in \bar{X}_1$, $\exists x_{2j} \in \bar{X}_2 : x_{2j} \to_h x_{1i}$. If we take the inverse h^{-1} of h then it will map all variables of Q_1 to some variables of Q_2. Since there exists also an onto containment mapping $h': Q_1 \to Q_2$ we have that the inverse h^{-1} will map all variables of Q_1 to all variables of Q_2 such that all subgoals of Q_2 have a mapping from all subgoals of Q_1. Hence Q_1 and Q_2 are isomorphic.

The following Theorem is now a result again of Definition 16 and this time of Theorem 5. Checking isomorphism between two CQs is GI-complete where GI is the complexity of checking graph isomorphism, which belongs in NP but is not proven to be either in P or NP- complete [Mat79].

Theorem 7. *Given two conjunctive queries Q_1 and Q_2 we have that the following are equivalent:*
i) $Q_1 \simeq Q_2$, i.e., Q_1 and Q_2 are isomorphic.
ii) $Q_1 \equiv_{TR} Q_2$.
iii) $Q_1 \equiv_{SBase} Q_2$.
iv) $Q_1 \equiv_{Same} Q_2$.
Checking conjunctive query equivalence under TR, SBase or Same-lineage semantics has Graph-Isomorphism (GI)-complete complexity.

From Theorem 7 it follows that the possible LDB instances of two TR, SBase or Same isomorphic queries posed over a ULDB will have equal data and lineage. Hence if redundant joins of a CQ are removed, the result will have different lineage that will affect TR, SBase and Same lineage equivalence. From the definition of Same-lineage equivalence semantics we have that this result also holds for database equivalence, i.e., the possible instances of two Same equivalent ULDBs will have equal data and lineage. On the other hand this does not hold for TR and SBase ULDB database containment. For example consider LDBs D and D' of Figure 3.18, where we have both $D \subseteq_{TR} D'$, $D' \subseteq_{TR} D$ and $D \subseteq_{SBase} D'$, $D' \subseteq_{SBase} D$, but the lineage functions of D and D' are not equal (for example we have $11 \in \lambda'(31)$ while $11 \notin \lambda(31)$ - only $11 \in \lambda^*(31)$ holds). As a result TR and SBase Lineage ULDB database containment are not partial orders.

3.9 UCQ Queries Containment and Equivalence

We now extend our results to unions of conjunctive queries (UCQs).

Theorem 8. *Given two unions of conjunctive queries $\bar{Q}_1 = \{Q_{11}, Q_{12}, \ldots, Q_{1n}\}$ and $\bar{Q}_2 = \{Q_{21}, \ldots, Q_{2m}\}$ we have that the following are equivalent:*
i) $\bar{Q}_1 \subseteq_{Data} \bar{Q}_2$.
ii) $\bar{Q}_1 \subseteq_{CBase} \bar{Q}_2$.
iii) for every Q_{1i}, $i = [1 \ldots n]$ there exists a $Q_{2j} \in \bar{Q}_2$ and a containment mapping (not necessarily onto)
$h: Q_{2j} \rightarrow Q_{1i}$.
The query containment test for UCQs with the above semantics is NP-complete.

Proof. Let U be an arbitrary ULDB and $\bar{Q}_1 = \{Q_{11}, Q_{12}, \ldots, Q_{1n}\}$, $\bar{Q}_2 = \{Q_{21}, Q_{22}, \ldots, Q_{2m}\}$ two UCQs. Suppose that $\bar{Q}_1(U) \subseteq_{CBase} \bar{Q}_2(U)$ holds. From Proposition 1 it follows that we have an one-to-one correspondence between the possible instances of $\bar{Q}_1(U)$ and $\bar{Q}_2(U)$. Let D_k be a possible instance of U. Let D_{k1} be the possible instance of $\bar{Q}_1(U)$ that contains the answer of $\bar{Q}_1$ posed on D_k and let D_{k2} be the corresponding possible instance of $\bar{Q}_2(U)$. When computing UCQs over ULDBs we have that a tuple with data t and lineage $\lambda(t)$ in $\bar{Q}_1(D_k)$ is the answer of a CQ $Q_{1i} \in \bar{Q}_1$. Similarly a tuple with data t and lineage $\lambda'(t)$ in $\bar{Q}_2(D_k)$ is the answer of a CQ $Q_{2j} \in \bar{Q}_2$. From this fact and Theorem 3 we have that if a tuple with data t and base lineage $\lambda_B(t)$ exists in $\bar{Q}_{1i}(D_k)$ then a tuple with the same data t and base lineage $\lambda'_B(t) \subseteq \lambda_B(t)$ exists in $\bar{Q}_{2j}(D_k)$ if and only if we have that for every Q_{1i}, $i = [1 \ldots n]$ there exists a $Q_{2j} \in \bar{Q}_2$ and a containment mapping (not necessarily onto) $h: \ Q_{2j} \rightarrow Q_{1i}$. Identifiers of relations R_{Q1} and R_{Q2} do not belong to S_- and tuples of relations in D_k remain with the same identifiers in D_{k1} and D_{K2}. Thus we have that $D_{k1} \subseteq_{CBase} D_{k2}$ (and $D_{k1} \subseteq_{Data} D_{k2}$) if and only if we have that for every Q_{1i}, $i = [1 \ldots n]$ there exists a $Q_{2j} \in \bar{Q}_2$ and a containment mapping (not necessarily onto) $h: \ Q_{2j} \rightarrow Q_{1i}$. □

In the above proof if we replace $\subseteq_{Data}$ or $\subseteq_{CBase}$ with one of TR, SBase or Same lineage query containment semantics then from Theorem 5 we get the following result:

Theorem 9. *Given two unions of conjunctive queries $\bar{Q}_1 = \{Q_{11}, Q_{12}, \ldots, Q_{1n}\}$ and $\bar{Q}_2 = \{Q_{21}, \ldots, Q_{2m}\}$ we have that the following are equivalent:*
i) for every Q_{1i}, $i = [1 \ldots n]$ there exists a $Q_{2j} \in \bar{Q}_2$ and an onto *containment mapping $h: \ Q_{2j} \rightarrow Q_{1i}$.*
ii) $\bar{Q}_1 \subseteq_{TR} \bar{Q}_2$.
iii) $\bar{Q}_1 \subseteq_{SBase} \bar{Q}_2$.
iv) $\bar{Q}_1 \subseteq_{Same} \bar{Q}_2$.
The query containment test for UCQs with the above semantics is NP-complete.

From Definition 16 which defines query equivalence as a two-way query containment the previous two theorems and Corollary 1 the next two theorems follow:

Theorem 10. *Given two unions of conjunctive queries $\bar{Q}_1 = \{Q_{11}, Q_{12}, \ldots,$ $Q_{1n}\}$ and $\bar{Q}_2 = \{Q_{21}, \ldots, Q_{2m}\}$ we have that the following are equivalent:*
i) for every Q_{1i}, $i = [1 \ldots n]$ there exists a $Q_{2j} \in \bar{Q}_2$ and a (not necessarily onto) containment mapping $h: \ Q_{2j} \rightarrow Q_{1i}$ and
for every Q_{2i}, $i = [1 \ldots m]$ there exists a $Q_{1j} \in \bar{Q}_1$ and a (not necessarily onto) containment mapping $h: \ Q_{1j} \rightarrow Q_{2i}$

ii) $\bar{Q}_1 \equiv_{Data} \bar{Q}_2$.
iii) $\bar{Q}_1 \equiv_{CBase} \bar{Q}_2$.
Moreover the UCQ equivalent test for the above semantics is NP-complete.

Theorem 11. *Given two unions of conjunctive queries* $\bar{Q}_1 = \{Q_{11}, Q_{12}, \ldots, Q_{1n}\}$ *and* $\bar{Q}_2 = \{Q_{21}, \ldots, Q_{2m}\}$ *we have that the following are equivalent:*
i) for every Q_{1i}, $i = [1 \ldots n]$ *there exists a* $Q_{2j} \in \bar{Q}_2$ *such that there exists an onto containment mapping from* Q_{2j} *to* Q_{1i} *and*
for every Q_{2i}, $i = [1 \ldots m]$ *there exists a* $Q_{1j} \in \bar{Q}_1$ *such that there exists an onto containment mapping from* Q_{1j} *to* Q_{2i}
ii) $\bar{Q}_1 \equiv_{TR} \bar{Q}_2$.
iii) $\bar{Q}_1 \equiv_{SBase} \bar{Q}_2$.
iv) $\bar{Q}_1 \equiv_{Same} \bar{Q}_2$.
Moreover the UCQ equivalent test for the above semantics is GI-complete.

We note that both ways onto containment between two UCQs $\bar{Q}_1 = \{Q_{11}, \ldots, Q_{1n}\}$ and $\bar{Q}_2 = \{Q_{21}, \ldots, Q_{2m}\}$ does not yield that every Q_{1i} of $\bar{Q}_1$ will be isomorphic with a Q_{2j} of $\bar{Q}_2$, but the semantics of equivalence hold. We give an example to illustrate this subtlety:

Example 15. Consider UCQs $\bar{Q}_1 = \{Q_{11}, Q_{12}\}$ and $\bar{Q}_2 = \{Q_{21}, Q_{22}\}$ where:
$Q_{11}(x)$:-$Drives(x, y), Saw(z, y), Saw(w, y), Saw(v, y)$,
$Q_{12}(x)$:-$Drives(x, y), Saw(z, y), Saw(w, y)$,
$Q_{21}(x)$:-$Drives(x, y), Saw(z, y), Saw(w, y), Saw(v, y)$,
$Q_{22}(x)$:-$Drives(x, y), Saw(z, y)$.

We have that there exists an onto containment mapping from Q_{21} to both Q_{11} and Q_{12}. In addition there exists an onto containment mapping from from Q_{12} to both Q_{21} and Q_{22}. As a result condition (i) of Theorem 11 holds. We have that Q_{12} is not equivalent with none of the CQs of $\bar{Q}_2$ but UCQ equivalence holds: Obviously Q_{11} and Q_{21} are isomorphic since they are equal. So for tuples with data t and lineage $\lambda(t)$ occurring in $\bar{Q}_1$ from the answer of Q_{11} there will exist in $\bar{Q}_2$ a tuple with same data and lineage. For tuples occurring in $\bar{Q}_1$ from the answer of Q_{12} we have: There exists an onto containment mapping from Q_{11} to Q_{12} and as a result a tuple with data t and lineage $\lambda(t)$ occurring in the answer of Q_{12} will also exist in the answer of Q_{11} with the same data and lineage. Hence it will also exist in $\bar{Q}_2$ due to Q_{21}. The other direction will similarly hold for the data and lineage in the answer of $\bar{Q}_2$.

3.10 Other Definitions of ULDB Database Containment

In this Section we point the following interesting observation: We can define ULDB database containment for Data, CBase, TR, SBase and Base lineage using one of the two following alternative definitions. From Proposition 1 it follows that even when we define ULDB database containment using any of the following definitions instead of Definition 14, the results of Theorems 3− 11 still hold.

Definition 17. Let $\subseteq_L$ denote one of the variants $\subseteq_{Data}$, $\subseteq_{CBase}$, $\subseteq_{TR}$, $\subseteq_{SBase}$, $\subseteq_{Same}$ of LDB database containment of Definitions 12 or 13.
Let U and U' be two ULDB's. We say that U is ULDB L-contained in U' (denoted with $\subseteq_L$) if the following hold:
i) If U has n possible instances $D_1, D_2, \ldots, D_n$ then U' has the same number n of possible instances $D'_1, \ldots, D'_n$ and
ii) There exists an one-to-one and onto correspondence for all the possible instances of U and U', i.e., for each $i = [1 \ldots n]$ we have that $D_i \subseteq_L D'_i$ holds according to the corresponding notion of LDB containment definition.

Definition 18. Let U and U' be two ULDB's. We say that U is ULDB L-contained in U' (denoted with $\subseteq_L$) if the following holds:
For every possible instance D_i of U there exists a possible instance D'_j of U' such that: $D_i \subseteq_L D'_j$ holds according to the corresponding notion of LDB containment definition.

3.11 Uncertain Equality containment - $\subseteq_E$ and semantics of ULDB Equality containment

In [ASUW10] another kind of containment for uncertain databases with no lineage was discussed that was suitable for uncertain data integration purposes. An uncertain database (with no lineage and without identifiers) is defined as a set of ordinary databases called *Possible Worlds* (PWs) (in convention we use the notion *Possible Worlds* whenever we have no lineage and we retain the notion of *Possible Instances* in the case with lineage). The definition of equality containment is the following:

Definition 19 (Equality Containment $\subseteq_E$)**.** [ASUW10]
Base case for databases with one relation: Consider two uncertain databases U_1 and U_2 both containing a single relation R. Let $\mathrm{T(U_1)}$ denote the set of

all tuples appearing in any possible world of U_1 and respectively we define $T(U_2)$. We say that U_1 is equality-contained in U_2 ($U_1 \subseteq_E U_2$), if and only if: $T(U_1) \subseteq T(U_2)$ and $PW(U_1) = \{W \cap T(U_1) \mid W \in PW(U_2)\}$.
For databases with more than one relations: Consider two uncertain databases U_1 and U_2 that contain a same set of relations $\bar{R} = \{R_1, \dots R_n\}$. Then $U_1 \subseteq_E U_2$ holds if for every relation R_i of $\bar{R}$ we have that U_1 and U_2 restricted to relation R_i are equality contained according to the base case definition.

Informally the above containment means that if we throw away from the possible worlds of U_2 all tuples that do not appear in any possible world of U_1, then the resulting possible worlds are the worlds of U_1. The notions of database containment we defined in the previous sections were data-driven: all kinds of them implied set containment of data (Semantics #1). In contrast the notion of equality containment focuses on retaining correlation and mutual exclusion between data: if two tuples in the contained database occur only in different worlds then no containing world can include both of them. In addition if two tuples occur only together in a possible world of the contained database then any containing world that has one of them must have the other as well. We note that U_2 can have more PIs than U_1. In [ASUW10] another notion of superset containment was also discussed which coincides with equality containment for query containment. Thus wee do not discuss it in here.

Now consider an uncertain database U that is a set of n ordinary databases that are its possible worlds. Posing a query Q over it will give a new uncertain relation with n possible worlds, each containing the answer of Q over the possible worlds of U. We have that even an onto containment between two conjunctive queries, which yielded the stronger same-lineage conjunctive queries containment does not suffice for equality containment. We illustrate this in the following Example 16. This result is not surprising since the notion of equality containment does not focus on data containment like the ones we presented.

Example 16. Consider conjunctive queries $Q_1(x)$:- $R(x,x)$ and $Q_2(x)$:$-R(x,y)$. There exists an onto containment mapping from Q_2 to Q_1. This implies that Q_1 is Same-Lineage contained in Q_2. However we shall show that there exists a ULDB U such that Q_1 is not equality contained in Q_2. Consider an uncertain database U containing the uncertain relation $R = \{\{(a,a)\}, \{(b,b)\}, \{(a,b),(b,a)\}\}$, where a and b are two different constants. Then $Q_1(U) = \{\,\{a\}, \{b\}, \emptyset\,\}$ and $Q_2(U) = \{\,\{a\}, \{b\}, \{a,b\}\,\}$. If we restrict the third possible world of $Q_2(U)$ to the tuples a and b that occur in $Q_1(U)$ then the resulting world $\{a,b\}$ is not a possible world of $Q_1(U)$. Hence Q_1 is not equality contained in Q_2.

It turns out that two conjunctive queries Q_1 and Q_2 are equality contained if and only if they are homomorphically equivalent, i.e., there exists a containment mapping h: $Q_2 \rightarrow Q_1$ and a containment mapping h': $Q_1 \rightarrow Q_2$, or if Q_1 is the empty query:

Theorem 12. *Given two conjunctive queries Q_1 and Q_2 we have that $Q_1 \subseteq_E Q_2$ iff Q_1 is the empty query or there exists a containment mapping h: $Q_2 \rightarrow Q_1$ and a containment mapping h': $Q_1 \rightarrow Q_2$. In addition checking whether $Q_1 \subseteq_E Q_2$ is NP-complete.*

Proof. (if:) Let U be an arbitrary uncertain database with $D_1, \ldots, D_n$ possible worlds.

If Q_1 is the empty query then for every $i = 1 \ldots n$. we will have that $Q_1(D_i)$ is empty. Thus the set of all tuples appearing in any $Q_1(D_i)$ is the empty set. Hence regardless of the form of query Q_2 if we restrict the tuples of every $Q_2(D_i)$ to the empty set we will get empty worlds, which are equal with the empty possible worlds of $Q_1(D_i)$.

Suppose that Q_1 and Q_2 are homomorphically equivalent. For each D_i we will have that $Q_1(D_i)$ and $Q_2(D_i)$ have exactly the same data. As a result we will restrict the tuples of every $Q_2(D_i)$ to its own tuples. Thus the result will be the same world $Q_2(D_i)$ which is equal with the world $Q_1(D_i)$. So equality-containment will hold for every uncertain database U.

(only if:) Let U be an arbitrary uncertain database with $D_1, \ldots, D_n$ possible worlds. Suppose that $Q_1 \subseteq_E Q_2$ holds. So if we restrict all the possible worlds of $Q_2(U)$ to all the tuples appearing in any possible worlds of $Q_1(U)$, we will get the possible worlds of $Q_1(U)$. One way for this to hold is to have that Q_1 is the empty query. Otherwise if Q_1 is not the empty query we will suppose that Q_1 and Q_2 are not homomorphically equivalent and yield to the contradiction that $Q_1 \subseteq_E Q_2$ does not hold. We have three possible cases:

• If there exists only a homomorphism $h : Q_2 \rightarrow Q_1$ (and no homomorphism $h' : Q_1 \rightarrow Q_2$) : Example 16 is an example where there exists only a homomorphism $h : Q_2 \rightarrow Q_1$ but $Q_1 \subseteq_E Q_2$ does not hold.

• If there exists only a homomorphism $h' : Q_1 \rightarrow Q_2$ (and no homomorphism $h : Q_2 \rightarrow Q_1$): The equality-containment must hold for any uncertain database U. So let us suppose that U has only one arbitrary possible world D_1. Since there exists only a homomorphism $h' : Q_1 \rightarrow Q_2$ we have that $Q_2(D_1) \subseteq Q_1(D_1)$. So all tuples of $Q_2(D_1)$ will also be tuples of $Q_1(D_1)$, hence we do not throw away any tuple from $Q_2(D_1)$. From $Q_2(D_1) \subseteq Q_1(D_1)$ we have the possible world of Q_2 posed over D_1 is not equal with the possible world of Q_1 posed over D_1 and $Q_1 \subseteq_E Q_2$ does not hold.

• If there exists no homomorphism $h : Q_2 \rightarrow Q_1$ and no homomorphism $h' : Q_1 \rightarrow Q_2$: The equality-containment must hold for any uncertain database U. So let us suppose that U has two possible worlds D_{C1} and D_{C2}. Let us create a unique constant for each variable appearing in conjunctive queries Q_1 and Q_2. Suppose that D_{C1} is the canonical database of Q_1, i.e., all the subgoals of Q_1 with the chosen constants substituted for variables. Similarly suppose that D_{C2} is the canonical database of Q_2. Then $Q_1(D_{C1})$ contains the head of Q_1 with the chosen constants substituted for variables ("frozen head" of Q_1). Because there exists no homomorphism $h : Q_2 \rightarrow Q_1$ we have that $Q_2(D_{C1})$ does not contain the frozen head of Q_1. Because there exists no homomorphism $h' : Q_1 \rightarrow Q_2$ we also have that $Q_2(D_{C2})$ does not contain the frozen head of Q_1. As a result none of the two possible worlds of $Q_2(U)$ contains the frozen head of Q_1. So none of the two possible worlds of $Q_2(U)$ when restricted to tuples appearing in the possible worlds of $Q_1(U)$ can be equal to $Q_1(D_{C1})$. Hence $Q_1 \subseteq_E Q_2$ does not hold.

The NP-completeness follows from the completeness of checking equivalence between two queries over ordinary databases which holds if and only if the two queries are homomorphically equivalent [CM77]. □

Equality containment for ULDBs

In Sections 4,5,6 we presented five different notions of ULDB containment that concerned containment of data and of different kinds of lineage information. Equality containment for uncertain databases focuses on preserving the correlation of data in the possible worlds. We can retain this property by defining Equality Data (EData) containment $\subseteq_{EData}$ for ULDBs: Given two ULDBs U_1 and U_2 equality ULDB database containment will retain correlations of data between the possible instances of U_1 and U_2. EData containment is an extension of Equality containment Definition 19 for uncertain databases for the case of ULDBs. Since a ULDB apart from data and uncertainty also contains lineage and identifiers we can pose lineage constraints on top of the data constraints of EData equality ULDB containment. It is natural to consider the useful lineage constraints that we had in CBase, TR, SBase and Same lineage containment. We denote these four notions of ULDB equality containment with $\subseteq_{ECBase}$, $\subseteq_{ETR}$, $\subseteq_{ESBase}$, $\subseteq_{ESame}$. In the next subsection we show that introducing lineage constraints on top of Equality Data ULDB containment can be useful in a setting of ULDB data integration.

Given a set of data tuples T_1 and and LDB D we define the " T_1-Reduced" LDB $D_R^{T_1}$ as the LDB that we have when we remove from LDB D all tuples that have data not occurring in the set T_1, along with their identifiers and lineage. The LDB $D_R^{T_1}$ contains only tuples with data occurring in T_1 along with the original identifiers and lineage they had in LDB D.

A possible instance of a ULDB is an LDB which contains data, identifiers and lineage. Suppose we have two ULDBs U_1 and U_2 and $T(U_1)$ is the set of tuples appearing in any possible instance of U_1. In order extend Equality containment of Definition19 to ULDBs it is natural to require that if we reduce to $T(U_1)$ the PIs of U_2 then we will get PIs that have the same set of data tuples with the PIs of U_1. In other words as we saw in Section 3.8 we must have that for every possible instance D_1 of U_1 there exists a possible instance D_2 of U_2 and such that $D_1 \equiv_{Data} D_{2R}^{T(U_1)}$ and for every possible instance D'_2 of U_2 there exists a possible instance D'_1 of U_1 such that $D'_1 \equiv_{Data} D_{2R}'^{T(U_1)}$. We can now give Equality Data ULDB containment definition which concerns ULDBs with a single relation due to simplicity. Extending it for databases with many relations is straightforward similarly with Definition 19.

Definition 20 (Equality Data ULDB Database Containment $\subseteq_{EData}$)**.** Consider two ULDBs U_1 and U_2. Let $T(U_1)$ denote the set of all tuples (only their data without identifiers or lineage) appearing in any possible instance of U_1 and respectively we define $T(U_2)$. We say that U_1 is Equality Data ULDB contained in U_2 ($U_1 \subseteq_{EData} U_2$), if and only if:
1. $T(U_1) \subseteq T(U_2)$ and
2. for every possible instance D_1 of U_1 there exists a possible instance D_2 of U_2 such that $D_1 \equiv_{Data} D_{2R}^{T(U_1)}$ and
3. for every possible instance D_2 of U_2 there exists a possible instance D_1 of U_1 such that $D_1 \equiv_{Data} D_{2R}^{T(U_1)}$.

The answer of a query Q_1 over a ULDB U includes all ULDB relations of U and a new ULDB relation R_{Q1}. Thus relations of ULDB U exist in both $Q_1(U)$ and $Q_2(U)$, so we are interested in equality containment between the new relations R_{Q1} and R_{Q2}.

Apart from requiring that the possible instances of a ULDB U_1 will have the same set of tuples with the reduced to $T(U_1)$ set of tuples of the possible instances of U_1, we can additionally require that if a tuple with data t occurs with lineage λ in a possible instance of U_1, then a tuple with same data and lineage also occurs in a reduced to $T(U_1)$ possible instance of U_2. This lineage requirement gives rise to ESame ULDB equality containment semantics. We have the following definition:

Definition 21 (Equality Same ULDB Database Containment $\subseteq_{ESame}$)**.** Consider two ULDBs U_1 and U_2. Let T(U$_1$) denote the set of all tuples (only their data without identifiers or lineage) appearing in any possible instance of U_1 and respectively we define $T(U_2)$. We say that U_1 is Equality Same ULDB contained in U_2 ($U_1 \subseteq_{ESame} U_2$), if and only if:
1. $T(U_1) \subseteq T(U_2)$ and

2. for every possible instance $D1$ of U_1 there exists a possible instance $D2$ of U_2 such that $D1 \equiv_{Data} D2_R^{T(U_1)}$ and $D1 \subseteq_{Same} D2_R^{T(U_1)}$ and
3. for every possible instance $D2$ of U_2 there exists a possible instance $D1$ of U_1 such that $D1 \equiv_{Data} D2_R^{T(U_1)}$ and $D1 \subseteq_{Same} D2_R^{T(U_1)}$.

Similarly we define ECBase, ETR and ESBase equality ULDB containment by replacing $\subseteq_{Same}$ in the above definition with one of the corresponding notions of lineage containment in the cases where we require that for every tuple with data t existing in a PI of U_1 there exists in the reduced to $T(U_1)$ PIs of U_2 a tuple with same data and CBase, TR or SBase-contained lineage.

In contrast with Data, CBase, TR, SBase and Same lineage containment, for all kinds of equality containment we can have that query containment between two queries holds for LDBs but ULDB query containment does not hold for the same two queries. This is due to the fact that data existing in different possible instances can affect EData (and all other equality semantics) ULDB database and query containment. In such a case we can have LDB containment between all the PIs of two ULDBs U and U' but ULDB containment between U and U' may not hold (only the opposite holds). For LDBs EData CQ query containment coincides with ordinary query containment, i.e., $Q_1 \subseteq_{EData-LDB} Q_2$ if there exists a containment mapping h from Q_2 to Q_1: If we have an arbitrary LDB D, then, due to h, relation R_{Q2} of $Q_2(U)$ will have all the tuples occurring in the sets of tuples appearing in R_{Q1} and also possibly some more tuples which would not belong to the set $T(Q_1(D))$. As a result they will not occur in $Q2_R^{T(Q1(D))}$ and hence R_{Q2} reduced to tuples appearing in $Q_1(D)$ will have the same set of tuples with $Q_1(D)$, i.e., $Q_1 \subseteq_{EData} Q_2$ will hold for LDBs.

On the other hand this result does not hold for ULDB EData CQ query containment: Consider conjunctive queries $Q_1(x){:}{-}R(x,x)$ and $Q_2(x){:}{-}\ R\ (x,y)$. There exists a containment mapping from Q_2 to Q_1. This implies that Q_1 is LDB EData contained in Q_2. However we shall show that there exists a ULDB U such that Q_1 is not ULDB EData contained in Q_2. ULDBs are a complete model so they can represent any given set of possible instances. The ULDB shown in Figure 3.24 represents the three possible worlds of Example 16. This ULDB has three possible instances one for the three choices of mutually exclusive external identifiers $(31,1)$, $(31,2)$ and $(31,3)$. In the possible instance of R_{Q1} which occurs from selection of $(31,3)$ we have that R_{Q1} is empty and R_{Q2} contains two tuples, one with data a and one with data b. But tuples with data a and b occur in the (other two) possible instances of R_{Q1} so they will not be removed, similarly with Example 16. Hence $Q_1 \subseteq_{EData} Q_2$ does not hold for ULDBs even though it holds for LDBs.

ID	R
11	a,a || b,b || a,b
12	b,a '?'

$$\lambda(11,1) = \{(31,1)\}$$
$$\lambda(11,2) = \{(31,2)\}$$
$$\lambda(11,3) = \{(31,3)\}$$
$$\lambda(12,1) = \{(31,3)\}$$

Figure 3.24: ULDB representing Example 16.

Similarly for LDB containment ESame coincides with Same query containment, i.e., $Q_1 \subseteq_{ESame-LDB} Q_2$ if there exists an onto containment mapping from Q_2 to Q_1: Due to onto containment the answers of Q_1 and Q_2 will have the same sets of data and if a tuple with data t and lineage λ occurs in relation R_{Q1} of LDB relation $Q_1(U)$ then a tuple with same data and lineage will occur in LDB relation R_{Q2}. Hence $Q_1 \subseteq_{ESame} Q_2$ will hold for LDBs and weaker ECBase, ETR and ESame query containment for LDBs will hold as well. In contrast this requirement is not enough even for EData ULDB CQ query containment whose data requirement is subsumed in all other semantics: For queries $Q_1(x){:}{-}R(x,x)$ and $Q_2(x){:}{-}R(x,y)$ there exists an onto containment mapping from Q_2 to Q_1. So they are ESame CQ query contained for LDBs, but as we saw they are not EData, hence not ESame, CQ query contained for ULDBs. We will show in the last subsection of this Section that for ULDB conjunctive query containment under all five equality containment semantics we must have that there exists a (not necessarily onto) containment mapping from Q_1 to Q_2 as well.

The Usefulness of ULDB Equality Containment in Data Integration

Equality containment is shown in [ASUW10] to be useful for the problem of data integration of uncertain sources: Specifically it is important when each source has access to a part of information of an existing uncertain database. The problem of data integration is to find the "best" uncertain database that reconciles in the best way the hidden uncertain database from the information of the sources in order to answer queries.

In this setting of uncertain data integration there exists one implicit logical mediated uncertain database (*Mediator*) and each source is defined as a view over the Mediator (Local-As-View uncertain data integration). The semantics of uncertain data integration are given in the following two definitions of Views and Sources. These definitions in [ASUW10] concern equality containment. We naturally extend them for the five semantics of equality ULDB containment. A view under uncertain integration is defined from a *view extension* (an uncertain database V) and from a *view definition* Q which is posed over the Mediator in order to get the database V.

Consider a ULDB V, a query Q and let E_i be one of EData, ECBase, ETR, ESBase, ESame equality ULDB containment. For a (logical) ULDB M the ULDB uncertain database V is a *view extension* under Ei with respect to *view definition* Q if and only if $V \subseteq_{E_i} Q(M)$. A *source* $S = (V, Q)$ is specified by a view extension V and a view definition Q. View V contains the uncertainty, data and lineage in the source and Q is the query used to map the source to the Mediator. A key notion for uncertain and ULDB data integration is "consistency". Let E_i be one of EData, ECBase, ETR, ESBase, ESame equality ULDB containment. The set of sources $S = \{S_1 \ldots, S_m\}$, where $S_i = (V_i, Q_i)$, is *consistent* if and only if there exists a ULDB database (Mediator) M such that: i) $M \neq \emptyset$, and ii) $\forall_{i \in \{1,\ldots,m\}} Vi \subseteq_{E_i} Q_i(M)$.

A set of ULDB sources is consistent if and only if there exists a ULDB mediator from which they could have been derived. Checking consistency of sources is fundamental for uncertain data integration: Query answering is meaningful only if a given set of views is consistent. In [ASUW10] it was shown that for uncertain databases without lineage and for equality containment $\subseteq_E$ the complexity of checking consistency is NP-hard even when we have only identity views, i.e., for every source S_i we have that Q_i is the identity query.

When the uncertain sources are ULDBs and thus apart from data and uncertainty have also lineage information it is natural to consider equality ULDB containment semantics that also concern lineage. In ULDB data integration each source has access only to a part of the data of a ULDB Mediator M. As a result it is natural to have in M a tuple with data t and lineage λ_1 and another tuple again with data t this time with lineage λ_2, but a source to have access only to the first tuple. So a source that has a tuple with data t and lineage λ_1 should be ESame contained to Mediator M since data correlations in PIs required in EData containment also hold. This is why in ESame Definition 21 we required containment and not equality between the PIs of the contained ULDB (source) and the reduced PIs of the containing ULDB (mediator). On the other hand in ESame definition we required EData equivalence in order for sets of data to be equal, thus preserving data correlations. The same holds for all other three kinds of ULDB equality containment that concern lineage. We show in the following Example 17 why linage information, in our Example ESame lineage, can be important in a data integration scenario with ULDBs and simple EData containment does not suffice:

Example 17. Suppose we are given two views V_1 and V_2 with data the ULDB shown in Figures 3.26 and 3.27. Both views are defined through the identity query. The ULDBs V_1 contains two ULDB relations $Saw(witness)$ with names of witnesses and $Suspects(name)$ with names of $Suspects$. Lineage in $Suspects$ connects names of suspects with the witness that accused them. In addition due to lineage $\lambda(22,1) = \{(21,1)\}$ we have that $Cathy$ and Amy always coexist in a PI (this can represent for example the fact that they were witnesses of the same crime).

ULDB V_1 has two possible instances, occurring from the two choices of alternatives of x-tuple 21. The first possible instance of V_1 has in LDB relation Saw witnesses $Cathy$ and Amy and in LDB relation $Suspects$ $Kate$ because she was accused from $Cathy$. The second possible instance of V_1 has $Celine$ in Saw and $John$ in $Suspects$. ULDB V_2 has also two possible instances, occurring from the two choices of alternatives of x-tuple 21. The first possible instance of V_2 has in LDB relation $Suspects$ $Kate$ and $George$ because they were both accused from $Cathy$. The second possible instance of V_1 has $Celine$ in Saw and the emptyset in $Suspects$.

It is easy to check that V_1 and V_2 are consistent under ESame ULDB equality containment (hence under all other 4 kinds of ULDB containment): Specifically for the ULDB M shown in Figure 3.25 we have that: $M \neq \emptyset$, $V_1 \subseteq_{ESame} M$ and $V_2 \subseteq_{ESame} M$ (note that Q_1 and Q_2 are the identity queries). Suppose now that instead of V_2 we had another identical view V_2' shown in Figure 3.28. Then V_1 and V_2' are EData consistent but not ESame consistent. The reason is that in the first possible instance of V_2' LDB relation Saw has witnesses $Cathy$ and Amy and LDB relation $Suspects$ has $George$ with lineage pointing to Amy and not to $Cathy$. Thus we have $V_2' \subseteq_{EData} M$ but $V_2' \not\subseteq_{ESame} M$. But different lineage can be important: If for Example $Cathy$ was considered an unreliable witness and hence $George$ was deleted in M but not in V_2' (because witness Amy remained reliable) then not even EData consistency would hold. As a result M in this case would not be a suitable Mediator, while another M' with a tuple with data $George$ in $Suspects$ pointing to Amy would be ESame contained to V_1 and V_2'. The semantics of $ECBase$, ETR, $ESBase$ are also useful in data integration scenarios with reliable and unreliable data similar with the cases where corresponding notions of $CBase$, TR and $SBase$ database lineage containment were important.

CQ Query Complexity for CBase, EData, ETR, ESBase and ESame ULDB equality containment

ID	**Saw**(witness)
21	Cathy || Celine
22	Amy ‘?’

$\lambda(22, 1) = \{(21, 1)\}$

ID	**Suspects**(name)
31	John || Kate
32	George

$\lambda(31, 1) = \{(21, 2)\}$
$\lambda(31, 2) = \{(21, 1)\}$
$\lambda(32, 1) = \{(21, 1)\}$

Figure 3.25: Actual ULDB database M.

ID	**Saw**(witness)
21	Cathy || Celine
22	Amy ‘?’

$\lambda(22, 1) = \{(21, 1)\}$

ID	**Suspects**(name)
31	John || Kate

$\lambda(31, 1) = \{(21, 2)\}$
$\lambda(31, 2) = \{(21, 1)\}$

Figure 3.26: View V_1 (with Q_1 the identity query)

ID	**Saw**(witness)
21	Cathy || Celine
22	Amy ‘?’

$\lambda(22, 1) = \{(21, 1)\}$

ID	**Suspects**(name)
31	Kate
32	George

$\lambda(31, 1) = \{(21, 1)\}$
$\lambda(32, 1) = \{(21, 1)\}$

Figure 3.27: View V_2 (with Q_2 the identity query)

ID	**Saw**(witness)
21	Cathy || Celine
22	Amy ‘?’

$\lambda(22,1) = \{(21,1)\}$

ID	**Suspects**(name)
31	John
32	George

$\lambda(31,1) = \{(21,2)\}$
$\lambda(32,1) = \{(22,1)\}$

Figure 3.28: View V_2' (with Q_2' the identity query)

Even when we care only about preserving data correlations in possible LDB instances of a ULDB and we do not care about lineage or identifiers the following Theorem shows that $Q_1 \subseteq_{EData} Q_2$ if and only if there exists a containment mapping h from Q_2 to Q_1 and a containment mapping h' from Q_1 to Q_2.

Theorem 13. *Given two conjunctive queries Q_1 and Q_2 we have that* $Q_1 \subseteq_{EData} Q_2$
iff Q_1 is the empty query or there exists a containment mapping h: $Q_2 \rightarrow Q_1$ and a containment mapping h': $Q_1 \rightarrow Q_2$. In addition checking whether $Q_1 \subseteq_{EData} Q_2$ is NP-complete.

Proof. The case of an empty Q_1 is similar with the proof of Theorem 12.
(if:) Let U be an arbitrary ULDB with $D_1, \ldots, D_n$ possible instances. Suppose that there exists a containment mapping h: $Q_2 \rightarrow Q_1$ and a containment mapping h': $Q_1 \rightarrow Q_2$. Then for every possible instance D_i of U, LDB relation R_{Q1} of $Q_1(D_i)$ has the same set of tuples with LDB relation R_{Q2} of $Q_2(D_i)$. Since this holds for every D_i we have that the set of tuples $T(Q_1)$ occurring in all possible instances of R_{Q1} is equal with $T(Q_2)$. As a result for each possible LDB instance D_i' of ULDB relation R_{Q2} we have that $D'i_R^{T(Q1)} = D_i'$. Since there exists a containment mapping h: $Q_2 \rightarrow Q_1$ and a containment mapping h': $Q_1 \rightarrow Q_2$ we also have that $D_i' \equiv_{Data} D_i$. As a result we have that: $D'i_R^{T(Q1)} = D_i' \equiv_{Data} D_i$. Hence from Definition 20 we have that $Q_1 \subseteq_{EData} Q_2$.
(only if:) Suppose that $Q_1 \subseteq_{EData} Q_2$ holds. Let U be a ULDB representing possible instances with no lineage and no duplicates. From the fact that we have EData equality containment it follows from Definition 20 that the possible instances of $Q_2(U)$ containing only tuples occurring in any possible

instance of $Q_1(U)$ are equal with the possible instances of $Q_1(U)$. Since U represents possible instances with no lineage and no duplicates we have from Theorem 12 that there exists a containment mapping h: $Q_2 \rightarrow Q_1$ and a containment mapping h': $Q_1 \rightarrow Q_2$. $\square$

For all remaining notions of ECBase, ETR, ESBase and ESame equivalent conjunctive query containment we have that their definition always require EData containment. From Theorem 13 this holds if and only if there exists a containment mapping h from Q_2 to Q_1 and a containment mapping h' from Q_1 to Q_2. This means that all possible instances of relations R_{Q1} and R_{Q2} will have exactly the same sets of tuples and hence we do not remove any tuple from any possible instance of R_{Q2} when we compute its reduced LDB instances. As a result the condition: $D1 \subseteq_{ESame} D2_R^{T(U_1)}$ of Definition 21 becomes equivalent with: $D1 \subseteq_{ESame} D2$. Hence from the previous Theorem and from Theorems 3 and 5, the following two Theorems follow:

Theorem 14. *Given two conjunctive queries Q_1 and Q_2 we have that*
$Q_1 \subseteq_{ECBase} Q_2$
iff Q_1 is the empty query or there exists a containment mapping h: $Q_2 \rightarrow Q_1$ and a containment mapping h': $Q_1 \rightarrow Q_2$. In addition checking whether $Q_1 \subseteq_{ECBase} Q_2$ is NP-complete.

Theorem 15. *Given two conjunctive queries Q_1 and Q_2 we have that*
$Q_1 \subseteq_{ETR,ESBase,ESame} Q_2$
iff Q_1 is the empty query or there exists an onto containment mapping h: $Q_2 \rightarrow Q_1$ and a (not necessarily onto) containment mapping h': $Q_1 \rightarrow Q_2$. In addition CQ containment checking for these semantics is NP-complete.

Chapter 4

The Complexity of Data Exchange under Lineage and Uncertainty

We present a data exchange framework that is capable of exchanging uncertain data with lineage and give meaningful certain answers on queries posed on the target schema. The data are stored in a *database with uncertainty and lineage (ULDB)* which represents a set of possible instances that are *databases with lineage (LDBs)*. Hence we need first to revisit all the notions related to data exchange for the case of LDBs. Producing all possible instances of a ULDB, like the semantics of certain answers would indicate, is exponential. We present a more efficient approach: a u-chase algorithm that extends the known chase procedure of traditional data exchange and show that it can be used to correctly compute certain answers for conjunctive queries in PTIME for a set of weakly acyclic tuple generating dependencies. We further show that if we allow equality generating dependencies in the set of constraints then computing certain answers for conjunctive queries becomes NP-hard.

4.1 Introduction

Data exchange is the problem of translating data that is described in a source schema to a different target schema. The relation between source and target schemas is typically defined by schema mappings. Recently the data exchange problem has been widely investigated, even for various uncertain frameworks, e.g., probabilistic [FKK10]. One aspect of the problem is finding procedures that compute in polynomial-time target instances that represent adequate (usually for query answering purposes) information. A challenging problem that has received considerable attention is the problem of giving meaningful semantics and computing answers of queries posed on the target schema of a data exchange setting [ALP08a, ALP08b, FKK10]. In [FKMP05] it was shown that computing certain answers for conjunctive queries over ordinary relational databases can be done with polynomial data complexity if the constraints satisfy specific conditions (are *weakly acyclic*). Query answering for data exchange was also studied for aggregate queries [AK08] and queries with arithmetic comparisons [ALP08b]. To the best of our knowledge the data exchange problem and query answering has not been studied for models of databases that incorporate both uncertainty and lineage. On Uncertainty-Lineage Databases (ULDBs) an uncertain database represents a set of *possible instances (PIs)* that are certain databases with lineage (LDBs). The semantics for possible instances of an uncertain instance are that only one of them "captures the truth", but we do not know which. Thus we have two kinds of uncertainty: i) uncertainty about the possible instances that this source represents (which one is the "true") and ii) uncertainty that arises due to heterogeneous source and data schemas. Many modern applications like data extraction from the web, scientific databases, sensors and even data exchange systems of certain sources often contain uncertain data. These applications may require recording the origin of data, which is modeled through *lineage* or *provenance*. Databases that track down the provenance of certain data have been extensively studied [BKT01, CW03, GKT07].

The Trio model [BSH+08a] combines and supports both lineage and uncertainty in ULDBs. This model was proven to be complete for succinctly representing any set of databases with lineage (LDBs) [BSH+08a]. In this model a) each uncertain tuple (x-tuple) is a set of traditional tuples, called "alternatives", that represent its possible values (i.e. we have uncertainty as to which of them is the "true" one) and b) each alternative comes with its lineage. One of the reasons ULDB model was introduced is that it would be important for data exchange [BSH+08a]. Computing conjunctive queries, that is SPJ queries, on ULDBs was shown to require polynomial-time [BSH+08a]. Trio implements ULDBs in a database system build on top of a traditional SQL system with necessary encoding [BSH+08a].

In general a data exchange problem for a data model consists of a source instance represented in this model and of a set Σ of constraints that consists of source-to-target constraints Σ_{st} and target constraints Σ_t ($\Sigma = \Sigma_{st} \cup \Sigma_t$). Given a finite source instance I, the data exchange problem is to find a finite target instance J such that $< I, J >= I \cup J$ satisfies Σ_{st} and J satisfies Σ_t. Such a J is called *a solution for* I or, simply a *solution* if the source instance I can be easily understood [FKMP05]. The query answering problem on a data exchange setting is which target instance to materialize to be used for obtaining meaningful answers (called *certain answers*) of a query. In this chapter we investigate the *data exchange* problem for the ULDB model. We consider source-to-target (s-t) and target tuple-generating dependencies (tgds) as constraints. In addition we consider a ULDB source whose lineage is "well-behaved", a constraint defined in [BSH$^+$08a] and in practice is true for most databases. The important property of "well-behaved" lineage for our results is that the lineage of a tuple, even if expanded, cannot refer to itself (lineage transitive closure does not contain cycles).

We investigate the data exchange problem of the Trio model and address the relevant query answering problem for Conjunctive Queries (CQs). Our contributions are: i) We present natural semantics for ULDB certain answers. ii) We give *u-chase* algorithm which extends the well-known chase algorithm for certain databases. iii) We use u-chase to show that if the source is a well-behaved ULDB, the source to target constraints consists of tgds and the target of a set of weakly acyclic tgds then we can compute ULDB certain answers in PTIME. iv) We prove that if we incorporate equality generating dependencies (egds) in the set of target dependencies then computing certain answers for conjunctive queries on a ULDB data exchange setting becomes NP-hard.

Intuitively the semantics of ULDB certain answers are the following: A certain answer is a ULDB that represents a set of possible LDB instances. We want those instances to be the same as if we had considered first the LDB possible instances of the source ULDB and computed certain answers for each data exchange problem that rose from each one possible LDB source instance. An obvious way to tackle this problem is to follow the direction that semantics indicate: if the ULDB source has n possible LDB instances then produce n LDB certain answers computed from n LDB data exchange problems. But the number of the possible instances of a (source) ULDB can be exponential on the size of the data [BSH$^+$08a]. As a result this procedure would be computationally very expensive and would not be suitable for large data sets. In contrast our proposed u-chase algorithm computes certain answers in polynomial time for weakly acyclic tgds. Another reason which shows that moving from the well-studied data exchange problem for ordinary databases to databases with uncertainty and lineage is not trivial is the following: In the former case computing certain answers with egds in the target dependencies remains polynomial, while in our case we prove that this problem becomes NP-hard. The correct meaning of *ULDB certain answer* semantics is illustrated in Example 18. For simplicity it does not include any target constraints.

Example 18. Suppose that a local police department contains a database **A** with two relations: `Saw(witness,car)` and `Drives(person,car)` (we have borrowed the source schema from [BSH+08a]). The department has a list of drivers along with their cars that contains the information that *Hank* drives a *Honda*, *Jimmy* a *Mazda* and *Billy* a *Toyota* car. All this information is certain and stored in relation `Drives(person,car)`. The source ULDB relation `Drives` is shown in Figure 4.1. For every crime if a witness reports that he/she saw a car near the crime scene then the local department stores in relation `Saw` the `witness` and the `car`-make that the witness saw. Suppose now that a witness named *Cathy* saw a car near the crime scene but she was not sure whether it was a *Honda* or a *Mazda*. A witness named *Amy* saw a car but she was uncertain whether it was a *Toyota* or a *Honda*. Note that in Figure 4.1 empty lineage of the source data is omitted.

A ULDB relation consists of x-tuples instead of tuples. Each x-tuple has a unique identifier and a multiset of "alternative values", separated with || symbol. The semantics are that only one alternative (or zero if we have symbol '?') from each x-tuple can be true in a possible instance [BSH+08a]. For example ULDB relation *Saw* on Figure 4.1 has four LDB possible instances due to the two choices of alternatives for x-tuple 11 and the two choices for 12. If *Saw* had n x-tuples each with two alternatives and again empty lineage then it would represent 2^n possible instances (exponential in the number of alternatives). As we will see in the following section lineage will pose further logical complex constraints to the possible instances that a UDLB relation represents.

Now a private investigator owns a database **B** that has a single relation: `Suspects&Dates(suspect,date)`. Suppose that the private investigator wants to transfer the information from the database of the local police department to his own database. But he only wants to store the `suspect` and `date` for each crime since he is not interested about witness' names or car information. The following source-to-target tgd ξ models this example: $\texttt{Saw}(\texttt{witness}, \texttt{car}), \texttt{Drives}(\texttt{p}, \texttt{car}) \rightarrow \exists D\, \texttt{Suspects\&Dates}(\texttt{p}, \texttt{D})$. We note that `Suspects&Dates` contains only the names of persons `p` that drove a `car` that was seen near a crime-scene (i.e., contains only suspects). The `date` attribute is not present in the source schema and is represented as an "unknown" - "null" value in the target. Even in certain data exchange the heterogeneity between the source and the target schema gave rise to "null" values that appear as distinct variables in a database instance [FKMP05]. Hence a materialized target instance that we expect due to tgd ξ is shown in Figure 4.2.

Suppose now that the private investigator wants to ask about all the names of all the suspects. This query posed over his target database **B** is the

ξ: $\texttt{Saw}(\texttt{witness}, \texttt{car}), \texttt{Drives}(\texttt{p}, \texttt{car}) \rightarrow \exists D\, \texttt{Suspects\&Dates}(\texttt{p}, \texttt{D})$
Q: $\texttt{Q}(\texttt{suspect}) : -\texttt{Suspects\&Dates}(\texttt{suspect}, \texttt{date})$

ID	**Saw**(witness,car)
11	Cathy,Honda || Cathy,Mazda
12	Amy,Toyota || Amy,Honda

ID	**Drives**(person,car)
21	Hank, Honda
22	Jimmy, Mazda
23	Billy, Toyota

Figure 4.1: ULDB **source** database **A**.

ID	**Suspects&Dates** (suspect,date)
31	Hank,d_1 ?
32	Jimmy,d_2 ?
33	Billy,d_3 ?
34	Hank,d_4 ?

$$\lambda_B(31) = \{(11,1),(21,1)\}$$
$$\lambda_B(32) = \{(11,2),(22,1)\}$$
$$\lambda_B(33) = \{(12,1),(23,1)\}$$
$$\lambda_B(34) = \{(12,2),(21,1)\}$$

Figure 4.2: Expected materialized **target** ULDB instance.

ID	**Q**(suspect)
41	Hank ?
42	Jimmy ?
43	Billy ?
44	Hank ?

$$\lambda_B(41) = \{(11,1),(21,1)\}$$
$$\lambda_B(42) = \{(11,2),(22,1)\}$$
$$\lambda_B(43) = \{(12,1),(23,1)\}$$
$$\lambda_B(44) = \{(12,2),(21,1)\}$$

Figure 4.3: Expected Certain Answer of Q posed on **target** database **B**.

following conjunctive query Q: `Q(suspect) :- Suspects&Dates(suspect, date)`. Query Q is a projection of attribute `suspect` of `Suspects&Dates`. Since the source instance is a database with uncertainty, the target database **B** will also be uncertain and have lineage pointing to source data. It will represent a set of *possible instances* which are databases with lineage (and no uncertainty). Accordingly the answer of the query Q posed on B will represent a set of possible instances. Intuitively we expect the following:

i) $Hank$ should exist as a suspect in the certain answers of Q if the car that $Cathy$ saw was in reality a $Honda$ car. So in the relation Q the lineage will record the pointer back to the fact that $Cathy$ saw a $Honda$ (base alternative $11, 1$). Answer $Hank$ also comes from the certain fact that $Hank$ drives a $Honda$ (alternative $21, 1$). ii) But $Hank$ can also be a suspect in the certain answers of Q if the car that *Amy* saw was in reality a $Honda$ car. Thus the suspect $Hank$ will be recorded twice in the answers but with different lineage. As we see in an LDB we can have that tuples with same data appear many times if their lineage is different. iii) Similarly we expect $Jimmy$ to exist as a suspect in the certain answers of Q if the car that $Cathy$ saw was in reality a $Mazda$ car, so with lineage that contradicts the lineage $(11, 1)$ of the first tuple with $Hank$ data (since at most one alternative can be true from x-tuple 11).

As a result we expect the certain answer of Q to be a ULDB as shown in Figure 4.3. With $s(i, j)$ we denote the j-th alternative of x-tuple with identifier i. *Base data* of a ULDB instance is its source data with empty lineage. If two alternatives point to the same base data we say that they have the same *base lineage* which is lineage extended back and containing only base data. In Figure, 4.3 $\lambda_B(s(i, j))$ denotes the *base lineage* of an alternative $s(i, j)$. We note that lineage does not just track where data comes from, but also poses logical restrictions to the possible LDB instances that a ULDB represents. An alternative of an x-tuple can be true in a possible instance if its lineage is true. For example Jimmy cannot appear in all possible instances but only to the two ones that have selected alternative $(11, 2)$ to be true. Symbol '?' indicate that there exists a possible instance that does not contain any alternative from this x-tuple (such a tuple is called a "maybe" x-tuple [BSH+08a]).

Note that in a certain data exchange setting where we have no lineage and no uncertainty tuple $(Hank, d_4)$ in materialized target instance Suspects&Dates is redundant since there already exists tuple $(Hank, d_1)$, so as concerns only data, tgd ξ is satisfied. But if we take lineage and uncertainty into account if we have only $(Hank, d_1)$ and not $(Hank, d_4)$ in our target then in the possible instance which has selected alternatives $(11, 2)$ $(Cathy, Mazda)$ and $(12, 2)$ $(Amy, Honda)$ we would have tgd ξ would require a tuple with `suspect` $Hank$ to appear in the target from tuples $(12, 2)$ and $(21, 1)$. But $(Hank, d_1)$ has base lineage $\{(11, 1), (12, 2)\}$ so it will not appear in the this possible instance which has selected $(11, 2)$ from the two mutually exclusive alternatives of x-tuple 11. As a result in a ULDB data exchange setting we also have to take lineage into account. Since the possible instances of a ULDB are determined from the choices of base data we need to retain tuples in the target instance that point back to different base data even though they have same data. This is why tuple $Hank, d_4$ must exist in the ULDB target instance even though $Hank, d_1$ already exists since they have different base lineage.

Other data models that represent a set of possible worlds to capture incomplete information are presented in [IL84]. The conditional tables of [IL84] are complete for representing possible instances that are certain databases but do not support lineage tracking. In [GKIT07] peer data exchange for LDBs is considered that uses mappings of various trustworthiness and focuses on filtering out the untrusted answers. In their lineage (provenance) model, they record more detailed information than the LDB model we consider in our work here.

4.1.1 Related Work

A similar with data exchange problem is data integration. In [SDH] data integration with uncertain mappings is considered but for certain sources. The procedure that is presented produces first all possible certain data integration problems that are represented due to uncertain mappings and then computes ordinary certain answers separately. In [MM09] the sources can be uncertain (but with no lineage) and several properties of uncertain data integration (a problem generally more complicated than data exchange) are formalized and discussed. The procedure that is used for certain query answering first produces all possible instances without uncertainty.

4.1.2 Ordinary Data Exchange

We refer to the data exchange problem for traditional databases with no lineage nor source uncertainty as *ordinary/certain data exchange problem.* Even in this case due to the fact that source and target can be different schemas we can have many possible target instances that satisfy the constraints called *solutions.*

Definition 22 (Certain Data Exchange)**.** [FKMP05]
A data exchange problem $(\mathbf{S}, \mathbf{T}, \mathbf{\Sigma_{st}}, \mathbf{\Sigma_t})$ consists of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, a set Σ_{st} of source-to-target dependencies, and a set Σ_t of target dependencies.
The data exchange problem associated with this setting is the following: given a finite source instance I, find a finite target instance J such that $< I, J >= I \cup J$ satisfies Σ_{st} and J satisfies Σ_t. Such a J is called a solution for I or, simply a solution if the source instance I is understood from the context. The set of all solutions for I is denoted by $Sol(I)$.

Definition 23 (Tuple Generating Dependency)**.** Let D be a database schema. A tuple generating dependency (tgd) is a first-order logical formula of the form:

$$\forall \mathbf{x} \big(\phi(\mathbf{x}) \to \exists \mathbf{y}\ \psi(\mathbf{x}, \mathbf{y})\big)$$

where $\phi(\mathbf{x})$ is a conjunction of atomic relational formulas over D with variables in $\mathbf{x}$. Each variable in $\mathbf{x}$ occurs in at least one formula in $\phi(\mathbf{x})$. In addition, $\psi(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic relational formulas over D with variables in $\mathbf{x}$ and $\mathbf{y}$ and each variable in $\mathbf{y}$ occurs in at least one formula in $\psi(\mathbf{x}, \mathbf{y})$.

In the rest of this chapter we will drop the universal quantifier in front of a tgd rule. Further we will refer to to the right hand side (left hand side respectively) of a tgd with *rhs* or *body* (*lhs* or *head* respectively). Satisfaction of a tgd for a traditional certain database is defined as:

Definition 24 (Tgd satisfaction)**.** Let d be a tgd of the form:

$$\forall \mathbf{x}\big(R_1(\mathbf{x_1}), R_2(\mathbf{x_2}), \ldots, R_n(\mathbf{x_n})\big) \to$$
$$\exists \mathbf{y}\big(T_1(\mathbf{x'_1}, \mathbf{y_1}), T_2(\mathbf{x'_2}, \mathbf{y_2}), \ldots, T_m(\mathbf{x'_m}, \mathbf{y_m})\big)$$

where $\mathbf{x_i} \in \mathbf{x}$, $\mathbf{y_i} \in \mathbf{y}$.
A database D will satisfy d if for every homomorphism h that maps $R_1(\mathbf{x_1}), R_2(\mathbf{x_2}), \ldots, R_n(\mathbf{x_n})$ to tuples $R_1(t_1), R_2(t_2), \ldots, R_n(t_n)$ in D then there exist a homomorphism h' that is an extension of h that maps the rhs of d to tuples $T_1(t'_1), T_2(t'_2), \ldots, T_n(t'_n)$ in D.

The following is an example of a certain data exchange problem.

Example 19. Let us modify our running Example 18 so that the source is an ordinary certain database with no uncertainty and without keeping track of the lineage of derived tuples. In particular we now suppose that Cathy and Amy are certain that they saw a Honda car. Relation Drives was already certain so it will stay the same. We only have to remove lineage and the unique IDs of each tuple that were used for lineage tracking. The certain source instance is now shown in Figure 4.4.

The source-to-target tgd of Example 18 stays the same. So we assume that again the private investigator wants to transfer the information from the database of the local police department to its own database, but he only wants to store the Suspects and Date for each crime:

ξ :$\texttt{Saw}(\texttt{i},\texttt{w},\texttt{c}),\texttt{Drives}(\texttt{p},\texttt{c}) \rightarrow$

$\quad\exists D$ $\texttt{Suspects\&Dates}(\texttt{i},\texttt{p},\texttt{D})$

According to Definition 24, Figure 4.5 shows a possible solution J_1 where d_1 represents an "unknown" null value that is represented through a unique variable).

The query Q_1 also remains the same, i.e. the investigator wants to ask his database for the names of all the suspects:

$\texttt{Q}_1(\texttt{suspect})$:- $\texttt{Suspects\&Dates}(\texttt{suspect},\texttt{date})$

Figure 4.6 shows the result of applying Q_1 to J. In this case our particular J contains no null values and $Q_1(J)$ coincides with the certain answers of Q_1 of a certain data exchange setting with I as source and ξ_1 as constraints. Intuitively regardless of the date of the crime, Hank will be a suspect.

Saw (witness, car)
Cathy, Honda
Amy, Honda

Drives (person, car)
Hank, Honda
Jimmy, Mazda
Billy, Toyota

Figure 4.4: Source certain instance I

We finish this section with the following standard definitions from certain data exchange:

Suspects&Dates(suspect,date)
Hank, d_1

Figure 4.5: Solution J for I and ξ

$Q_1(J)$
Hank

Figure 4.6: $Q_1(J)$

Definition 25 (Conjunctive Query - CQ)**.** A conjunctive query $Q(x)$ over a schema T is a formula of the form $\exists y\, \phi(x, y)$, where $\phi(x, y)$ is a conjunction of atomic formulas over T.

Definition 26 (Weakly acyclic tgds)**.** For a set Σ of tgd's over a database schema $\bar{R}$ the dependency graph of Σ is the directed graph that has as vertices (R, i), where $R \in \bar{R}$, and $i \in 1, \ldots, arity(R)$. The graph has two types of edges:
1. Regular edges. There is a regular edge between vertices (R, i) and (S, j) if there a tgd in Σ that has a variable y that appears both in position (R, i) in the body, and in position (S, j) in the head.
2. Existential edges. There is an existential edge between vertices (R, i) and (S, j) if there is a tgd in Σ that has a variable $x24$ that appears in position (R, i) of the body and in some position in the head, and an existentially quantified variable z that appears in position (S, j) in the head.

A set of Σ of tgd's is said to be *weakly acyclic* if the dependency graph of Σ does not have any cycles containing an existential edge.

Definition 27 (Ordinary Chase)**.** The ordinary chase works on an instance K with constants and null values. Let Σ be a set of tgd's and let ξ be a tgd in Σ. Let h be a homomorphism from the lhs of ξ to I. If there exists no homomorphism which is an extension of h that maps the rhs of ξ to K then an extension h'' of h which maps the existential variables of the rhs of ξ to fresh new null values is applied to the rhs of ξ producing new tuples that are added in K. In that case we say that the ordinary chase "fires" with ξ under h' producing a new instance K', denoted with $K \rightarrow^{\xi,h'} K'$.

In [FKMP05] the following was shown:

Theorem 16. *Let Σ be a weakly acyclic set of tgd's. Then ordinary chase terminates in polynomial time.*

4.2 Data Exchange with Lineage

We have that the *possible instances (PIs)* of a ULDB are LDBs [BSH+08a]. Since we do not know which one captures the "truth", intuitively all the LDB possible instances of a ULDB solution must satisfy data exchange constraints. So we need to revisit all the data exchange relevant definitions of the certain case for an LDB data exchange setting. The data exchange problem for databases with lineage has the same setting as in the certain case. The difference now is that the tuples in the source instance and the target instance (to be materialized) have lineage information. The LDB model extends the relational model in the way that every tuple apart from its data has also a unique identifier attached and a lineage function pointing to the set of tuple identifiers from which this tuple was derived from. As we already discussed in the Introduction of this Chapter, heterogeneity between the source and the target schema can give rise to "null" values that appear as distinct variables in a target instance.

We denote by $Const$ the set of all values that occur in source instances and we call them constants. In addition, we assume an infinite set Var of values, which we call *labeled nulls*, such that $Var \cap Const = \emptyset$. If $\bar{\mathbf{R}}$ is a database schema and K is an instance over $\bar{\mathbf{R}}$ with values in $Const \cap Var$, then $Var(K)$ denotes the set of labelled nulls occurring in relations in K.

When tuples have values from constants then we say that it is a *ground database instance with lineage (ground LDB instance)*. When tuples have values from constants and variables (where these two sets are disjoint) then we say that we have a *database instance with lineage (LDB instance)*.

The lineage function is empty for base relations. For derived relations it points to the identifiers of the tuples from which it was derived. If a tuple is derived as an answer of a query, lineage points to the tuples that were used in order to get this tuple in the answer relation. Algorithms 1 and 2 that we present on Chapter 2 compute conjunctive queries (CQs) over LDBs and ULDBs following the methods presented on [BSH+08a].

We focus on computing certain answers and correct lineage. Correct lineage for a tuple that is derived as an answer of a CQ has been already defined in CQ computing algorithms of [BSH+08a]. But now we also have to compute semantically correct lineage for a tuple that is derived from a data exchange setting. The important difference between LDB conjunctive query computing and CQ computing on ordinary databases is the following: Now a tuple can exist in the answer of a query more than once if it is derived from different tuples. So two or more tuples can have the same data if they have different lineage. In a similar way the semantics of satisfying tuple generating dependencies should also be slightly different. We will refer to the right hand side (left hand side respectively) of a tgd with *rhs* or *body* (*lhs* or *head* respectively). It is natural to define that a tuple generating dependency rule d is LDB-satisfied if it is satisfied in the traditional sense and, moreover, the lineage of the image of the lhs tgd atoms is related to the lineage of the image of the rhs tgd atoms. For Data Exchange purposes it turns out that we are interested in retaining that tuples come from the same base tuples, i.e., have the same set of base lineage. This is natural since the base tuples are tuples appearing in the source database instance and with same base lineage we make sure that the target tuples come from the correct "top-origin" of the source. The usefulness of such a definition is more obviously seen when we use it in the context of ULDB data exchange query answering. For now we only note that the base data is the one which defines: i) the number of possible instances of a ULDB and ii) which derived tuples with well-behaved lineage will be also appearing (apart from the base ones) at each instance. This property is proved in [BSH+08a]. So base data is the one which exclusively determines (through base lineage) the LDB possible instances of a ULDB. The formal definition follows:

Definition 28 (LDB satisfaction of a tgd)**.** Let D be an LDB and d be a tgd of the form:
$\forall \mathbf{x}\big(R_1(\mathbf{x_1}), R_2(\mathbf{x_2}), \ldots, R_n(\mathbf{x_n}) \rightarrow \exists \mathbf{y}\big(T_1(\mathbf{x'_1}, \mathbf{y_1}), T_2(\mathbf{x'_2}, \mathbf{y_2}), \ldots, T_m(\mathbf{x'_m}, \mathbf{y_m})\big)\big)$
We will say that LDB D l-satisfies d, if for each homomorphism h that maps $R_1(\mathbf{x})$, $R_2(\mathbf{x}), \ldots,$, $R_n(\mathbf{x})$ to tuples: $R_1(t_1)$ with $ID = ID_1$, $R_2(t_2)$ with $ID = ID_2$, …, $R_n(t_n)$ with $ID = ID_n$ in D, then there exists a homomorphism h' that is an extension of h that maps the rhs of d to tuples $T_1(t'_1), T_2(t'_2), \ldots, T_m(t'_n)$ in D with :
$\lambda_B(t'_1) = \{\lambda_B(ID_1) \cup \lambda_B(ID_2) \ldots \cup \lambda_B(ID_n)\}$,
$\lambda_B(t'_2) = \{\lambda_B(ID_1) \cup \lambda_B(ID_2) \ldots \cup \lambda_B(ID_n)\}$,
$\vdots$
$\lambda_B(t'_m) = \{\lambda_B(ID_1) \cup \lambda_B(ID_2) \ldots \cup \lambda_B(ID_n)\}$,
where, e.g., $\lambda_B(t'_1)$ is the base lineage of tuple t'_1. In the above union if a tuple with identifier ID_k is base, we replace $\lambda_B(ID_k)$ not with its empty lineage but with base identifier ID_k.

The identifiers in the answer of a CQ apart from the fact that need to be unique and different than the existing ones, their exact value is not important. For example a tuple with identifier 41, data value *Hank* and lineage $\lambda(41) = \{11, 21\}$ says that *Hank* exists in the answer due to facts with ids $\{11, 21\}$. At this point the newly created answer-identifier 41 could have been as well 51: the important fact is that *Hank* is in the answer and due to base tuples $\{11, 21\}$. We will have in mind this difference in order to define l-certain answers: For a tuple t^{LDB} of an LDB we can "drop" its identifier information and thus take a pair $\big(t,\ \lambda(ID(t))\big)$, which we denote by $ID_{drop}\ (t^{LDB})$. We extend ID_{drop} to apply to databases:

Definition 29. Given an LDB D, we define $ID_{drop}(D)$ to be derived from D by changing each tuple t^{LDB} in each relation of D to $ID_{drop}(t^{LDB})$.

Since it is the base lineage which determines the possible instances of a ULDB and is important in LDB tgd-satisfiability we are interested in retaining in LDB-certain answers LDB tuples with data and base lineage that must appear in every solution in order to satisfy the tgds.

Given an $ID_{drop}(t^{LDB})$ tuple we can replace its lineage with its base lineage and thus take a pair $\big(t, \lambda_B(ID(t)\big)$, which we denote with $\lambda_{base}(ID_{drop}$ $(t^{LDB}))$. We extend λ_{base} to apply to databases:

Definition 30. Given an LDB D, we define $\lambda_B(ID_{drop}(D))$ to be derived from D by changing each tuple t^{LDB} in each relation of D to $\lambda_{base}(ID_{drop}$ $(t^{LDB}))$.

Since we have no alternatives, LDB tuples are always present and their unique identifiers can be single numbers and not pairs referring to alternatives (like it is the case for ULDBs). When a tuple's identifier is clear from the context, we may refer to an LDB tuple only with its unique identifier ID (i.e. denote its lineage as $\lambda(ID)$). Alternatively, we may say that we want to have an LDB tuple with data t and lineage $\lambda(t)$ present in our database, when confusion does not arise.

We can extract only relations of target schema J using polynomial extraction algorithm found in [BSH+08a]. It extracts a subset of relations of an LDB or a ULDB along with all their base lineage information. Data from tuples not appearing in retained relations are not being kept. Only their lineage and only if it is referred through the base lineage of remaining tuples. From now on we use the term *extracting* a relation, which is applying the following extraction algorithm [BSH+08a] with the extracted relation as its input.

EXTRACTION ALGORITHM:

1. **input:** ULDB $D = (\bar{R}, S, \lambda)$, and $\bar{X} \subseteq \bar{R}$
2. **output:** a ULDB $D' = (\bar{X}, S', \lambda')$
3. $S' = I(\bar{X}) \cup (\bigcup_{x \in I(\bar{X})} \lambda^*(x))$
4. $\lambda' = \lambda|_{S'}$, the restriction of λ to S'

5. **return** D'

We can now define LDB certain answers and LDB solutions:

Definition 31 (LDB Solutions)**.** Let $(\mathbf{S}, \mathbf{T}, \mathbf{\Sigma_{st}}, \mathbf{\Sigma_t})$ be an LDB data exchange setting. An LDB *solution* to this data exchange setting is an LDB J of target schema $\mathbf{T}$ such that together with I l-satisfies the tgds in Σ.

Definition 32 (LDB Certain Answers)**.** Let q be a conjunctive query over the target schema T and I an LDB source instance. For each LDB solution J we drop the identifiers producing $ID_{drop}(q(J))$. We further replace all lineage with its base lineage producing $\lambda_{base}(ID_{drop}(q(J)))$ Then $l\text{-}cert_{drop}(q, I) = \cap\{\lambda_{base}(ID_{drop}(q(J)))\}$. Finally the LDB *certain answers* (l-certain answers) are produced by attaching a fresh distinct identifier to each tuple of $l\text{-}cert_{drop}(q, I)$ and produce $l\text{-}cert(q, I)$.

Universal solutions of a certain data exchange setting are target relations, such that there exists a homomorphism from them to each other solution [FKMP05]. In order to define LDB Universal Solutions we first have to revisit the notion of homomorphism. We keep the "data part" of definitions as it is in the certain case and for the lineage part we require the mapped tuples to have the same base lineage. Once again, the usefulness of such a definition is more obviously seen when we use it in the context of ULDB data exchange query answering where the base lineage will pose logical restrictions to the possible LDB instances that the ULDB Universal Solution represents, since base data is the one which exclusively determines (through base lineage) the LDB possible instances of a (well-behaved) ULDB [BSH+08a]. We can now define LDB homomorphism (denoted as l-homomorphism) and LDB Universal Solution.

Definition 33 (LDB homomorphism h_l)**.** Let D_1, D_2 be two LDB instances over the same schema with values in $Const \cup Var$. We say that D_1 l-maps to D_2 ($D_1 \rightarrow_l D_2$) if there exists a homomorphism h from variables and constants of D_1 to variables and constants of D_2 such that: i) it maps every base fact t_i^1 of D_1 to a base fact $h(t_i^1) = t_j^2$ of D_2 with $ID(h(t_i^1)) = ID(t_j^2)$ and ii) for every non base fact $R_i(t)$ of D_1 we have that $R_i(h(t))$ is a fact of D_2 with $\lambda_B(R_i(h(t)) = \lambda_B(R_i(t))$, where λ_B is the lineage extended back to and containing only base data.

Definition 34 (LDB Universal Solution)**.** Consider a data exchange setting $(\mathbf{S}, \mathbf{T}, \mathbf{\Sigma_{st}}, \mathbf{\Sigma_t})$. If I is a source LDB instance, then an l-universal solution for I is a solution J such that for every solution J' we have that there exists an LDB homomorphism h_l such that: $J \rightarrow_{h_l} J'$.

4.3 Computing LDB certain answers

Now that we have all the necessary notions from the previous sections, we can define l-chase. L-chase will be used in the next section in order to compute ULDB certain answers in PTIME. Again the "data part" of l-chase will be similar to the certain definition. Certain chase is a procedure that makes sure that its result will satisfy a given tgd. We will similarly also need to extend it with a "lineage part", according to our definition of l-homomorphism in order for its result to l-satisfy a tgd:

Definition 35 (l-chase step)**.** Let K be an LDB instance. Let d be a tgd of the form: $\forall \mathbf{x}\big(R_1(\mathbf{x_1}), R_2(\mathbf{x_2}), \ldots, R_n(\mathbf{x_n}) \rightarrow \exists \mathbf{y}\big(T_1(\mathbf{x'_1}, \mathbf{y_1}), T_2(\mathbf{x'_2}, \mathbf{y_2}), \ldots,$ $T_m(\mathbf{x'_m}, \mathbf{y_m})\big)\big)$. Let h be a homomorphism from $R_1(\mathbf{x}), R_2(\mathbf{x}), \ldots, R_n(\mathbf{x})$ to tuples $R_1(t_1)$, $R_2(t_2), \ldots, R_n(t_n)$ in K with IDs: $\{\text{ID}_1, \text{ID}_2, \ldots, \text{ID}_n\}$ such that there exist no homomorphism h' that is an extension of h that maps the rhs of d to tuples $T_1(t'_1), T_2(t'_2), \ldots, T_m(t'_m)$ in K with:
$\lambda_B(t'_1) = \lambda_B(t_1) \cup \lambda_B(t_2) \cup \ldots \cup \lambda_B(t_n)$,
$\lambda_B(t'_2) = \lambda_B(t_1) \cup \lambda_B(t_2) \cup \ldots \cup \lambda_B(t_n)$,
$\vdots$
$\lambda_B(t'_m) = \lambda_B(t_1) \cup \lambda_B(t_2) \cup \ldots \cup \lambda_B(t_n)$.
We say that d can be l-applied to K with homomorphism h (in this case l-chase "fires").

Let LDB K' be the union of K with the set of facts obtained by: (a) extending h to h' such that each variable in $\mathbf{y}$ is assigned a fresh labeled null, followed by (b) taking the image of the atoms of $\mathbf{y}$ under h' and adding them to the relations of the rhs of d along with a new non-used identifier (c) set as lineage of each atom of (b) the set $\{\text{ID}_1, \text{ID}_2, \ldots, \text{ID}_n\}$. We say that the result of applying d to K with h is K' , and write $K \xrightarrow{d,h}_l K'$.

Definition 36 (l-Chase)**.** Let Σ be a set of tgds and K be an LDB instance.
• An *l-chase sequence* of K with Σ is a sequence (finite or infinite) of l-chase steps $K_i \xrightarrow{d,h}_l K_{i+1}$, with $i = 0, 1, \ldots,$ with $K = K_0$ and d_i a dependency in Σ.
• A *finite l-chase* of K with Σ is a finite l-chase sequence $K_i \xrightarrow{d,h}_l K_{i+1}$, $0 \leq i < m$, where there is no dependency d_i of Σ and there is no homomorphism h_i such that d_i can be l-applied to K_m with h_i. The result of the finite l-chase is then K_m. In this case of a finite chase sequence we say that l-chase *terminates.*

With $\Lambda_B^{d,h}$ we denote the base lineage to which the lhd of d is pointing to under h, which is computed in the following way: $\Lambda_B^{d,h} = \Lambda_B(ID_1) \cup$

$\Lambda_B(ID_1), \cup \ldots \cup \Lambda_B(ID_n)$. The function $\Lambda_B(ID)$ maps the identifier of a base tuple to itself and the identifier of a non-base tuple to the set of identifiers in its base lineage.

Our l-chase is a sequence of l-chase steps. It is an extension of the well known chase procedure for traditional databases. Our l-chase algorithm is an extension of the well known chase procedure for traditional databases.

We also define parallel chase step $I \rightarrow_{\Sigma} I'$ of an instance I with a set of tgds Σ. Parallel chase step fires all applicable to I ordinary chase steps simultaneously:

Definition 37 (Parallel chase step)**.** Let Σ be a set of tgds and I be an instance consisting of constants and nulls. A parallel chase step $I \rightarrow^{Sigma} I'$ fires with all tgds ξ_i in Σ such that for each ξ_i there exists a homomorphism h from the lhs of ξ_i to I for which there exists no extension homomorphism h' that maps the rhs of ξ_i to I.

We can naturally define parallel l-chase step to fire over an LDB instance I with all tgds ξ_i in Σ such that for each ξ_i there exists a homomorphism h from the lhs of ξ_i to I for which there exists no extension homomorphism h' that maps the rhs of ξ_i to tuples of I with base lineage equal with the base lineage to which the lhs points to. We illustrate the differences of (parallel) l-chase and ordinary (parallel) chase in the following example:

Example 20. Consider an ordinary source instance I with two unary relations R_1 and R_2, with one tuple with data 1 in R_1 and one tuple with data 2 in R_2. Consider now the LDB source instance I' shown in Figure 4.7. Instance I' has two relations R_1 and R_2 with the same data with ordinary I. But an LDB can have many tuples pointing to different lineage. So let us suppose that LDB Relation R_1 has two tuples with same data 1 the first pointing to base lineage 11 and the second to base lineage 12. LDB Relation R_2 also has two tuples with same data 2 the first pointing to base lineage 13 and the second to base lineage 14. Base tuples $11, 12, 13, 14$ can be thought as external tuples or we can suppose that there exists also a source relation R with four LDB tuples with IDs $11, 12, 13, 14$.

Now consider the set of tgds $\Sigma = \{\xi_1, \xi_2, \xi_3, \xi_4\}$, where:
$\xi_1 \ : \ R_1(x) \rightarrow T_1(x)$
$\xi_2 \ : \ R_2(x) \rightarrow T_2(x)$
$\xi_3 \ : \ T_1(x), T_2(y) \rightarrow \exists z T_3(x, y, z)$
$\xi_4 \ : \ T_1(x), T_3(x, y, v) \rightarrow T_2(y)$
Ordinary parallel chase will terminate in two steps: The first parallel ordinary chase step will fire with ξ_1 and ξ_2 and create one tuple with data 1 in T_1 and one tuple with data 2 in T_2. The second parallel ordinary chase step will fire

ID	R_1
21	1
22	1

$\lambda_B(21) = \{11\}$
$\lambda_B(22) = \{12\}$

ID	R_2
31	2
32	2

$\lambda_B(31) = \{13\}$
$\lambda_B(32) = \{14\}$

Figure 4.7: I' Source LDB Instance of Example 20.

with ξ_3 and create tuple $1, 2, z_1$ in T_3. Then ξ_4 or any other tgd will not fire again.

Now parallel l-chase will terminate after four parallel l-steps: The first parallel l-chase step will fire with ξ_1 and ξ_2 and create two LDB tuples with data 1 in T_1, one pointing to base lineage $\{11\}$ and the other to $\{12\}$ and two LDB tuples with data 2 in T_2. The second parallel l-chase step will fire with ξ_3 and create tuples in T_3 shown in Figure 4.8. Then, in the third step, ξ_4 will fire and create T_2 shown in Figure 4.9. In the fourth and final step l-chase step will fire again with ξ_3 and create tuples in T_3 shown in Figure 4.10. We now have that l-chase terminates: L-chase will not fire again with ξ_4 over the new created tuples $45, 46, 47, 48$ of T_3.

E.g., if we map the lhs of ξ_4 to LDB tuple of T_3 with data $1, 2, z_5$ and $\lambda_B = \{11, 12, 13\}$ and to LDB tuple of T_1 with data 1 and $\lambda_B = \{11\}$, then we will have that there already exists tuple with data 2 in T_2 pointing to base lineage $\{11, 12, 13\} \cup 11 = \{11, 12, 13\}$. It is easy to check that the same holds for all possible mappings of the lhs of ξ_4.

In general we have that under any LDB instance parallel l-chase under Σ will fire at most in k' more parallel l-chase steps than ordinary parallel chase over an arbitrary instance, where k' is the constant number of target to target tgds in Σ, in our example $k' = 2$.

For the source to target tgds ξ_1 and ξ_2 chase and l-chase will both perform one parallel step: If there exists a tuple with data t and base lineage λ_{B1} in R_1 then l-chase will fire and add a tuple with data t with base lineage λ_{B1} in T_1 and ξ_1 will not fire again. The same holds for ξ_2 in the first l-chase parallel step.

Now parallel ordinary chase terminates in two parallel steps. We will show that l-chase terminates in at most more $k' = 2$ parallel l-chase steps. Since we have target to target tgds the reason for l-chase to fire another parallel l-chase step is when a newly created tuple in the rhs of a tgd can make another tgd to fire. This can only happen if an atom in the rhs of a tgd is repeated in the lhs of another (or the same) tgd. In our example T_2 in the rhs of ξ_4 is repeated in the lhs of ξ_3. In addition T_3 in the rhs of ξ_3 is repeated in the lhs of ξ_4. We want to show that after at most 2 more parallel l-chase steps l-chase will not fire again. Hence we must show that after $2 + 2 = 4$ parallel chase steps the newly creation of a tuple in T_2 from the rhs of ξ_4 will not cause ξ_3 to fire again and also that the newly creation of a tuple in T_3 from the rhs of ξ_3 will not cause ξ_4 to fire again.

Equivalently we can show that after at most $k = 4$ parallel l-chase steps for every tuple in T_2 (resp. T_3) with λ_{B2} to be created from an l-chase step there already exists a tuple in T_2 (resp. T_3) with λ_{B3} such that $\lambda_{B2} = \lambda_{B3}$ and such that their exist a homomorphism from the data of the one to be created to the one already existing.

Since we want to show that l-chase will stop after k' more parallel steps let us suppose that we have an arbitrary LDB instance J that has arbitrary LDB tuples in target relations T_1, T_2. So let an LDB tuple with data t_1 and base lineage λ_{B1} exist in T_1 and an LDB tuple with data t_2 and base lineage λ_{B2} exist in T_2. Since ordinary parallel chase has stopped, l-chase continues only if homomorphically equivalent tuples make tgds fire due to their lineage. In the second parallel l-chase step we have that ξ_3 fires and creates a tuple t'_3 in T_3 with base lineage $\lambda_{B1} \cup \lambda_{B2}$. In the third parallel l-chase step tuples t_1 and t_3 will make ξ_4 fire and create a tuple t'_2 in T_2 homomorphically equivalent to t_2 with base lineage $\lambda_{B1} \cup \lambda_{B3}$ and a tuple t''_2 in T_2 homomorphically equivalent to t_2 with base lineage $\lambda_{B1} \cup \lambda_{B2} \cup \lambda_{B3}$. In the fourth parallel l-chase step the newly created, homomorphically equivalent to t_2, tuple t''_2 of T_2 with base lineage $\lambda_{B1} \cup \lambda_{B2} \cup \lambda_{B3}$ will make ξ_3 to fire again with t_1 and t''_2 creating a tuple t''_3 in T_3 homomorphically equivalent to t_3 with base lineage $\lambda_{B1} \cup \lambda_{B2} \cup \lambda_{B3}$. Now for ξ_4 we have: The tuple t'_3 of T_3 created in the second parallel step has base lineage $\lambda_{B1} \cup \lambda_{B2} \cup \lambda_{B3}$. So there exists a homomorphism from the lhs of ξ_4 to tuples t_1 and t'_3 to a tuple homomorphically equivalent to t_2 with base lineage $\lambda_{B1} \cup \lambda_{B2} \cup \lambda_{B3}$. But T_2 already has a tuple homomorphically equivalent to t_2 with base lineage $\lambda_{B1} \cup \lambda_{B2} \cup \lambda_{B3}$. As a result ξ_4 will not fire again after the fourth l-parallel step. Hence l-chase will terminate after $k = 4$ parallel l-chase steps.

Definition 38 (l-derivation graph)**.** Let K be an LDB instance over a schema $\bar{R}$ consisting of relation atoms. Let Σ be a set of tgds of the form: $\phi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ \psi(\mathbf{x}, \mathbf{y}, \mathbf{z})$. We construct the *l*-derivation graph of K over Σ, denoted as

ID	T_3
41	1,2,z_1
42	1,2,z_2
43	1,2,z_3
44	1,2,z_4

$$\lambda_B(41) = \{11, 13\}$$
$$\lambda_B(42) = \{11, 14\}$$
$$\lambda_B(43) = \{12, 13\}$$
$$\lambda_B(44) = \{12, 14\}$$

Figure 4.8: Target LDB relation T_3 of Example 20 after second parallel l-chase step

ID	T_2
51	2
52	2
53	2
54	2
55	2
56	2
57	2
58	2
59	2
60	2

$$\lambda_B(51) = \{13\}$$
$$\lambda_B(52) = \{14\}$$
$$\lambda_B(53) = \{11, 13\}$$
$$\lambda_B(54) = \{11, 12, 13\}$$
$$\lambda_B(55) = \{11, 14\}$$
$$\lambda_B(56) = \{11, 12, 14\}$$
$$\lambda_B(57) = \{12, 13\}$$
$$\lambda_B(58) = \{11, 12, 13\}$$
$$\lambda_B(59) = \{12, 14\}$$
$$\lambda_B(60) = \{11, 12, 14\}$$

Figure 4.9: Target LDB relation T_2 of Example 20 after third parallel l-chase step.

ID	T_3
41	1,2,z_1
42	1,2,z_2
43	1,2,z_3
44	1,2,z_4
45	1,2,z_5
46	1,2,z_6
47	1,2,z_7
48	1,2,z_8

$$\lambda_B(41) = \{11, 13\}$$
$$\lambda_B(42) = \{11, 14\}$$
$$\lambda_B(43) = \{12, 13\}$$
$$\lambda_B(44) = \{12, 14\}$$
$$\lambda_B(45) = \{11, 12, 13\}$$
$$\lambda_B(46) = \{11, 12, 14\}$$
$$\lambda_B(47) = \{11, 12, 13\}$$
$$\lambda_B(48) = \{11, 12, 14\}$$

Figure 4.10: Target LDB relation T_3 of Example 20 after fourth and final parallel l-chase step

Gr^K_Σ, as follows:
• Each node of the tree is a pair $(R_i(t), \lambda_B)$ consisting of an LDB tuple with data t in relation $R_i \in \bar{R}$ and a set of base lineage λ_B.
• The bottom nodes of the *d*erivation tree of K over Σ consist of all pairs $(R_i(t), \lambda_B)$ where R_i is an LDB relation of K which has an LDB tuple with data t and pointing to base lineage λ_B.
• We then keep adding edges and new nodes by repeating the following procedure:
An edge connects a node $(R_i(t), \lambda_B)$ with a new node $(R'_i(t'), \lambda'_B)$ if there exists in Σ a tgd $\xi\ :\ \phi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ \psi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ such that:
i) $R_i \in \phi(\mathbf{x}, \mathbf{y})$ and $R'_i \in \psi(\mathbf{x}, \mathbf{y}, \mathbf{z})$.
ii) $(R_i(t), \lambda_B)$ is a node in Gr^K_Σ and there exists in Σ a tgd $\xi\ :\ \phi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\ \psi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ such that a homomorphism h maps the lhs of ξ to Gr^K_Σ, where atom $R_i \in \phi(\mathbf{x}, \mathbf{y})$ is mapped to an LDB tuple in relation R_i with data t and base lineage λ_B such that there exists no extension of h homomorphism h' which maps the rhs of ξ to existing nodes in Gr^K_Σ so that atom $R'_i \in \psi(\mathbf{x}, \mathbf{y})$ is mapped to an LDB tuple in relation R'_i with data t' and base lineage $\lambda'_B = \lambda_B$.

We add to Gr^K_Σ a node $R'_i(t'), \lambda'_B$ where t' is the tuple to which h' maps atom $R'_i \in \psi(\mathbf{x}, \mathbf{y}, \mathbf{x})$ and λ'_B is the union of the base lineages of all LDB tuples that are mapped from the lhd of ξ under h. We repeat the same procedure for the new Gr^K_Σ.

Consider for example LDB K with a single LDB relation $T_1(x, y)$ which consists of a single LDB tuple with data $(1, 2)$ and pointing to base lineage λ_{B1}. Let Σ consist of the following two tgds:
$\xi_1:\ T_1(x, y) \rightarrow T_2(x, y)$ and
$\xi_2:\ T_2(x, y) \rightarrow T_1(x, y)$.
Then Gr^K_Σ will have $T_1, (1, 2), \lambda_{B1}$ as a single bottom node and an edge connecting $T_1, (1, 2), \lambda_{B1}$ to $T_2, (1, 2), \lambda_{B1}$. Both ordinary chase and l-chase will stop after one (parallel) step which will map $T_1, (1, 2), \lambda_{B1}$ to $T_2, (1, 2), \lambda_{B1}$. It is not always the case such that ordinary chase and l-chase stop at the same parallel step, as we show in the following example.

Example 21. Consider an ordinary source instance I with three binary relations R_1, R_2 and R_3 with one tuple with data $(1, 1)$ in R_1, one tuple with data $(1, 2)$ in R_2 and one tuple with data $(1, 3)$ in R_3. Consider now the LDB source instance I' with the same data with ordinary I but with LDB tuples $R_1(1, 1)$, $R_2(1, 2)$ and $R_3(1, 3)$ pointing to base lineages λ_{B1}, λ_{B2} and λ_{B3} respectively.

Now consider the set of tgds $\Sigma = \{\xi_1, \xi_2, \xi_3, \xi_4\}$, where:
$\xi_1\ :\ R_1(x, y) \rightarrow T_1(x)$

$\xi_2 \;:\; R_2(x,y) \rightarrow T_2(x)$
$\xi_3 \;:\; R_3(x,y) \rightarrow T_3(x)$
$\xi_4 \;:\; T_1(x) \rightarrow T_2(x)$
$\xi_5 \;:\; T_2(x) \rightarrow T_3(x)$
$\xi_6 \;:\; T_3(x) \rightarrow T_1(x)$

The l-derivation graph $Gr_{\Sigma}^{I'}$ is shown if Figure 21.

Ordinary parallel chase will terminate in one step: The first parallel ordinary chase step will fire with ξ_1, ξ_2 and ξ_3 and create three tuples with data 1 in T_1, T_2 and T_3. Then no other tgd will fire again since all T_1, T_2 and T_3 have equal data.

Now parallel l-chase will terminate after at most three more parallel l-steps, because we may have after the first step LDB tuples with equal data, but they are not pointing to the same base lineage:

The first parallel l-chase step will fire with ξ_1 ξ_2 and ξ_3 and create thee LDB tuples with data 1 and pointing to base lineage λ_{B1} in T_1, to base lineage λ_{B2} in T_2 and to λ_{B3} in T_3.

For the second l-chase parallel step we have the following: The newly created LDB tuple $T_1(1)$ with λ_{B1} will cause ξ_4 to fire and create tuple $T_2(1)$ with λ_{B1}. Similarly from $T_2(1)$ with λ_{B2} and ξ_5 we will get tuple $T_3(1)$ with λ_{B2} and from $T_3(1)$ with λ_{B3} and ξ_6 we will get tuple $T_1(1)$ with λ_{B1}.

In the third l-chase parallel step we will similarly have tuples $T_3(1)$ with λ_{B1}, $T_1(1)$ with λ_{B2} and $T_2(1)$ with λ_{B3} created.

L-chase will terminate after the third parallel step because newly created data cause LDB tuples with existing data and base lineage to be fired created again.

Consider now the subset Σ' of Σ which does not include ξ_6.

The l-derivation graph $Gr_{\Sigma'}^{I'}$ is shown if Figure 21. Ordinary chase again stops after one parallel step but parallel l-chase terminates after two more steps: In the longest path ξ_4 and ξ_5 fire again with l-chase.

Theorem 17. *i) Given a set of tgds, L-chase terminates if and only if ordinary chase terminates. ii) Given a set of weakly acyclic tgds Σ, l-chase terminates on Σ in polynomial time.*

Proof. i)
(if:)
Let Σ be a set of tgds of the form: $\xi_i :\; \phi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z}\; \psi(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Suppose that ordinary chase terminates with Σ over any ordinary instance. Let D be an arbitrary ordinary instance over a relation schema $\bar{R}$. Let I be an LDB instance over $\bar{R}$ whose LDB tuples have the same data with the ordinary tuples of D. Since D is arbitrary, I is also arbitrary. We will show that

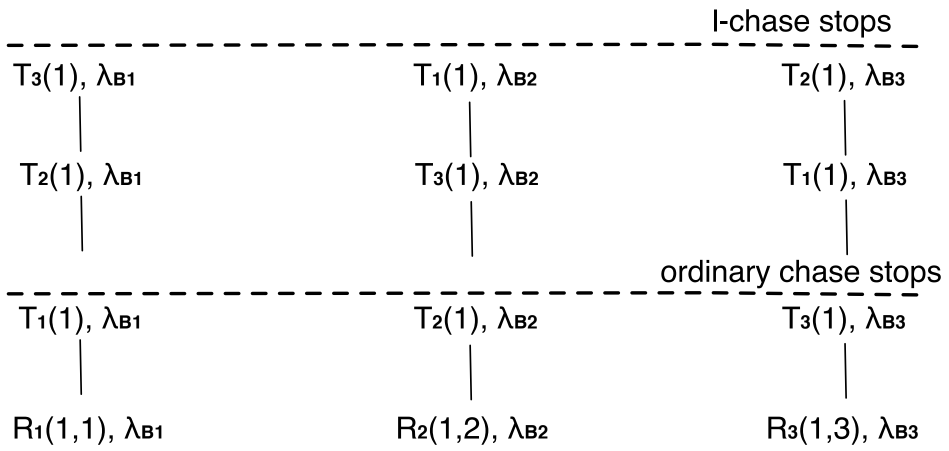


Figure 4.11: L-derivation graph $Gr_{\Sigma}^{I'}$ of Example 21.

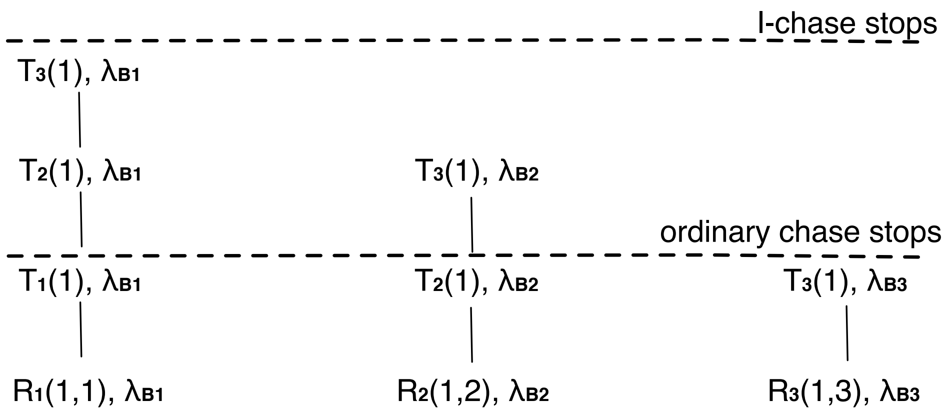


Figure 4.12: Finite L-derivation graph $Gr^{I'}_{\Sigma'}$ of Example 21.

l-chase terminates over I with Σ. Let Gr^I_Σ be the l-derivation graph of I over Σ. Since ordinary chase terminates we have that there exists a finite step k such that for each homomorphism h from the lhs of every $\xi_i \in \Sigma$ to an LDB tuple in R_i with data t there exists an extension of h homomorphism h' which maps the rhs of ξ_i to an LDB tuple in R'_i with data $h'(t)$.

Let $R_1(t_1), \lambda_{B1}$ be an arbitrary root node of Gr^I_Σ. From the definition 38 of l-derivation graph we have that when ordinary chase fires with a tgd, l-chase also fires and maps the lhs of the same tgd to Gr^I_Σ such an atom of its lhs is mapped to $R_1(t_1)$ then there exists a node in l-derivation graph Gr^I_Σ connecting $R_1(t_1), \lambda_{B1}$ to another node. We have that after at most k steps of ordinary chase in a path of Gr^I_Σ starting from root node $R_1(t_1), \lambda_{B1}$ we reach a node, say $T_2(t_2), \lambda_{B2}$, such that the data of all nodes of the path from $R_1(t_1), \lambda_{B1}$ to $T_2(t_2), \lambda_{B2}$ do not cause ordinary chase to fire again after the creation of node $T_2(t_2), \lambda_{B2}$. From the definition 38 of l-derivation graph we have that $\lambda_{B1} \subseteq \lambda_{B2}$.

If there exists no tgd in Σ with relation T_2 in its lhs then the arbitrary path in Gr^I_Σ starting from $R_1(t_1), \lambda_{B1}$ terminates in node $T_2(t_2), \lambda_{B2}$. From the definition of l-derivation graph we have that l-chase also does not fire again for the facts occurring in the nodes of this path and terminates in at most k steps.

If not let $\xi \ : \ \phi(\mathbf{x}, \mathbf{y}) \to \exists \mathbf{z} \ \psi(\mathbf{x}, \mathbf{y}, \mathbf{z})$ be a tgd in Σ such that $T_2 \in \phi(\mathbf{x}, \mathbf{y})$ and T_3 is an arbitrary atom of $\psi(\mathbf{x}, \mathbf{y}, \mathbf{z})$. Suppose that l-chase fires with ξ. So there exist in Gr^I_Σ an edge connecting $T_2(t_2), \lambda_{B2}$ to a node $T_3(t_3), \lambda_{B3}$. From the definition 38 of l-derivation graph we have that there exists a homomorphism h which maps the lhs of ξ to Gr^K_Σ, where atom $T_2 \in \phi(\mathbf{x}, \mathbf{y})$ is mapped to an LDB tuple in relation T_2 with data t_2 and base lineage λ_{B2} such that there exists no extension of h homomorphism h' which maps the rhs of ξ to existing nodes in Gr^K_Σ so that atom $T_3 \in \psi(\mathbf{x}, \mathbf{y})$ is mapped to an LDB tuple in relation T_3 with data t_3 and base lineage $\lambda_{B3} \subseteq \lambda_{B2}$.

Then in node $T_3(t_3), \lambda_{B3}$, we have that t_3 is the tuple to which h' maps atom $T_3 \in \psi(\mathbf{x}, \mathbf{y}, \mathbf{x})$ and the following hold: a)λ_{B3} is the union of the base lineages of the images of $\phi(\mathbf{x}, \mathbf{y})$ under h and, b) $\lambda_{B1} \subseteq \lambda_{B2} \subseteq \lambda_{B3}$.

Let K' be the LDB instance which includes facts with data and lineage occurring in nodes of Gr^I_Σ with distance less or equal than $k + k'$ where k is the number of steps where ordinary parallel chase stops and k' the number of target to target tgds. Tgd ξ fires with l-chase if there exists a homomorphism h from $\phi(\mathbf{x}, \mathbf{y})$ to K' such that there exists an extension h' which maps the $\psi(\mathbf{x}, \mathbf{y})$ to K' which maps an atom T_i in $\psi(\mathbf{x}, \mathbf{y})$ to a tuple with data t'_i in K' we have that the union of the base lineages of the images of $\phi(\mathbf{x}, \mathbf{y})$ in K' under h is not included in $\lambda_B(t'_i)$ of K'. But the union of the base lineages of the images of $\phi(\mathbf{x}, \mathbf{y})$ in K' under h already exists K' in node $T_3(t_3), \lambda_{B3}$. In addition if an atom in $\phi(\mathbf{x}, \mathbf{y})$ appears in the rhs of another tgd ξ' such that l-chase fires in a path starting from root node $R_1(t_1), \lambda_{B1}$ and going though nodes $T_2(t_2), \lambda_{B2}$ and $T_3(t_3), \lambda_{B3}$ then a fact with base lineage the image of $\phi(\mathbf{x}, \mathbf{y})$ from ξ' will also be created once for this rule after ordinary chase terminates.

As a result parallel l-chase will not fire again under ξ. Hence parallel l-chase can only fire at most once for every tgd after ordinary chase terminates, so at most k' times where k' is the number of target tgds in Σ. From the definition 38 of l-derivation graph we have that the arbitrary path starting from root node $R_1(t_1), \lambda_{B1}$ going though arbitrary node $T_2(t_2), \lambda_{B2}$ in at most k steps will reach a node that has no outgoing edges in at most more k' steps. Since the root node and the path where chosen arbitrarily, we have that any path starting from any root node of Gr^I_Σ will reach a node with no outgoing edges in at most $k+k'$ steps. Equivalently from the definition of l-derivation graph we have that parallel l-chase will terminate in at most $k + k'$ steps. Since parallel l-chase terminated we have that l-chase also terminates.

(only if:) Suppose that l-chases terminates with Σ. Let L_0 be an arbitrary LDB. We will show that ordinary chase terminates with Σ and any arbitrary ordinary instance. Let D_0 be the ordinary instance that has the same set of tuples with the set of the data that LDB tuples of arbitrary L_0 have. Since l-chase terminates we have on each l-chase step with ξ_i, h_i that there exists no extension of h_i such that it: a) maps the rhs of ξ_i to tuples in L_{i-1} and b) they point to the base lineage of the image of the lhs of ξ_i under h_i. Since (a) holds we have that the ordinary chase will also not fire. As a result the ordinary chase with D_0 and Σ also terminates (with fewer steps than l-chase).

ii) Let Σ be a set of k' weakly acyclic tgds. We have that ordinary chase terminates in polynomially many k steps. In the (if) part of the proof of (i) we have that parallel l-chase terminates in $k + k'$ steps. Since k' is a the constant number of tgds in Σ and k is polynomial to the size of data of any ordinary instance, we have that k is also polynomial to the size of the data of any LDB. As a result parallel l-chase terminates in polynomial time. Hence l-chase terminates in polynomial time.

□

Figure 4.14 shows the extraction of target relation *Suspects&Dates* from the result of l-chasing source LDB I with $\Sigma = \{\xi\}$. We are going to prove that the extraction of target relations from the result of a finite l-chase is always an l-universal solution.

Example 22. We give an example that is an "LDB version" of Example 19: Again we suppose that Cathy and Amy are certain that they saw a Honda car. Tuples now must have a unique ID since we will keep track of lineage so we will use the LDB data model. The LDB source instance, with omitted empty lineage for each tuple, is now shown in Figure 4.13.

The source-to-target tgd of Example 18 (or 19) is again the same:
$\xi : \texttt{Saw}(\texttt{w}, \texttt{c}), \texttt{Drives}(\texttt{p}, \texttt{c}) \rightarrow$
$\exists D$ $\texttt{Suspects\&Dates}(\texttt{p}, \texttt{D})$.

As we can see from Definition 32, Figure 4.14 shows a possible solution J_1 (again d_1 and d_2 represent "unknown" null values).

Indeed according to Definition 28 we have the homomorphism $h_{11} : w \rightarrow Cathy$, $c \rightarrow Honda$, $p \rightarrow Hank$ that maps the head $Saw(w, c), Drives(p, c)$ of ξ to tuples $Saw(Cathy, Honda)$ with ID=11 and $Drives(Hank, Honda)$ with ID=21 of LDB source instance I. Then it exists the extension h'_{11} which maps w $\rightarrow Cathy$, $c \rightarrow Honda, p \rightarrow Hank$ and $D \rightarrow d_1$ that maps the body $Suspects\&Dates(p, D)$ of ξ to tuple $Suspects\&Dates(Hank, d_1))$ of J that has lineage pointing to tuples with IDs 11 and 21.

Now for the homomorphism $h_{12} : w \rightarrow Amy, c \rightarrow Honda, p \rightarrow Hank$ that maps the head Drives(person, car) of ξ to tuples $Saw(Amy, Honda)$ with ID=12 and$Drives(Hank, Honda)$ with ID=21 of LDB source instance I we similarly have that it exists the extension h'_{12} which also maps $D \rightarrow d_2$ that maps the body $Suspects\&Dates(p, D)$ of ξ to tuple$Suspects\&Dates(Hank, d_2)$ of J that has lineage pointing to tuples 12 and 21.

The query Q_1 also remains the same, i.e. the investigator wants to ask his database for the names of all the suspects:
$\texttt{Q}_1(\texttt{Suspect})$:- $\texttt{Suspects\&Dates}(\texttt{Suspect}, \texttt{Date})$

Figure 4.15 shows the relation $R_{Q_1(J)}$ extracted from the result of applying Q_1 to J (using LDB query answering Algorithm 1, presented on Chapter 2).

Example 23. Continuing the previous example we have that intuitively regardless of the date of the crime, Hank will be a suspect. But infinitely many solutions can exist. In our example they are represented by the unknown values of d_1, d_2. As a result if the query posed on J was:
$\texttt{Q}_2(\texttt{s}, \texttt{d})$:- $\texttt{Suspects\&Dates}(\texttt{s}, \texttt{d})$
then we would take a different answer for each solution that has different d_1 or d_2 value. So a certain answer for Q_1 for our example coincides with the answer over J (because J has only constants) shown on Figure 4.15 while for Q_2 it is the empty set. We note that we could have another certain answer with different IDs, i.e. 51, 52 instead of 41,42. We would then require $\lambda_B(51) = \{11, 21\}$ and $\lambda_B(52) = \{12, 21\}$.

Figure 4.13: Source certain instance I

ID	Saw (witness, car)
11	Cathy, Honda
12	Amy, Honda

ID	Drives (person, car)
21	Hank, Honda
22	Jimmy, Mazda
23	Billy, Toyota

Figure 4.14: Solution J for I and ξ

ID	Suspects&Dates(person,date)
31	Hank, d_1
32	Hank, d_2

$$\lambda(31) = \{11, 21\}$$
$$\lambda(32) = \{12, 21\}$$

Figure 4.14 shows the extraction of target relation *Suspects&Dates* from the result of l-chasing source LDB I with $\Sigma = \{\xi\}$. We are going to prove that the extraction of target relations from the result of a finite l-chase is always an l-universal solution. The result of l-chase can be used in order to polynomially compute l-certain answers for weakly acyclic tgds as it is stated in the following Theorem:

Lemma 2. *Let K_2 be the result of an l-chase-step with an initial instance*

Figure 4.15: $Q_1(J)$

ID	$Q_1(J)$
41	Hank
42	Hank

$$\lambda(41) = \{31\}, \lambda_B(41) = \{11, 21\}$$
$$\lambda(42) = \{32\}, \lambda_B(42) = \{12, 21\}$$

K_1 and a tgd d ($K_1 \xrightarrow{d,h}_l K_2$). Let K be an LDB instance such that:
i) LDB K satisfies d and K contains K_1, that is for every LDB tuple with data t and lineage λ in K_1, there exists in K a tuple with same data and lineage.
ii) there exists an LDB homomorphism h_1: $K_1 \to K$.
Then there exists an LDB homomorphism h_2: $K_2 \to K$.

Proof. Let d be a tgd of the form:
$\forall \mathbf{x}, \mathbf{y} \big(R_1(\mathbf{x_1}, \mathbf{y_1}), R_2(\mathbf{x_2}, \mathbf{y_2}), \ldots, R_n(\mathbf{x_n}, \mathbf{y_1}) \to$
$\exists \mathbf{z} \big(T_1(\mathbf{x_1}, \mathbf{z_1}), T_2(\mathbf{x_2}, \mathbf{z_2}), \ldots, T_m(\mathbf{x_m}, \mathbf{z_m}) \big) \big)$

We have:
i) $K_1 \xrightarrow{d,h}_l K_2$:
From the Definition 35 of the l-chase step, let h be a homomorphism from the lhs of d, $R_1(\mathbf{x_1}), R_2(\mathbf{x_2}), \ldots, R_n(\mathbf{x_n})$ to LDB tuples with data $t_1, t_2, \ldots t_n$ with identifiers $ID_1, ID_2, \ldots ID_n$.

With $\Lambda_B^{d,h}$ we denote the base lineage to which the lhd of d is pointing to under h, which is computed in the following way: $\Lambda_B^{d,h} = \Lambda_B(ID_1) \cup \Lambda_B(ID_1), \cup \ldots \cup \Lambda_B(ID_n)$. The function $\Lambda_B(ID)$ maps the identifier of a base tuple to itself and the identifier of a non-base tuple to the set of identifiers in its base lineage. If l-chase does not fire then $K_1 = K_2$ and since K contains K_1 it also contains K_2 and the result is trivial.

Else if there exists no extension h' of h that maps the rhs of d to tuples in K_1 all with base lineage contained in Λ_B then l-chase "fires" with d and h. The l-chase step will produce instance K_2 by adding to K_1 the LDB tuples are the images of the rhs of d under h' with data $t'_1, t'_2, \ldots t'_n$ and all with same base lineage equal with $\Lambda_B^{d,h} = \Lambda_B(ID_1) \cup \Lambda_B(ID_1), \cup \ldots \cup \Lambda_B(ID_n)$.

ii) h_1: $K_1 \to K$:
Now h_1 is an LDB homomorphism from $K_1 = (\bar{R}, S, \lambda)$ to $K = (\bar{R}', S', \lambda')$. As a result all the base data of K_1 will be mapped under h_1 to base data in K and with the same IDs. For all data of K_1 that is not base, we have that homomorphism h_1 maps them to facts in K such that they have the same base lineage (from ii). As a result all tuples of K_1 are mapped through h_1 to tuples of K so that i) base tuples of K_1 are mapped to base tuples of K with same IDs and ii) non base tuples of K_1 become non base tuples of K and have the same base lineage. So for every tuple $t_i \in K_1$ with identifier ID'_i and $\lambda(ID'_i) = \emptyset$ we will have that h_1 maps it to tuple $h_1(t_i) \in K$ with $\lambda(ID'_i) = \emptyset$. For every non base tuple of $t_j \in K_1$ (with $\lambda(ID_j) \neq \emptyset$) we will have that h_1 maps it to tuple $h_1(t_j) \in K$ with $\lambda_B(ID'_j) = \lambda_B(ID_j)$. Hence all LDB tuples with data $t_1, t_2, \ldots t_n$ which occur in K_1 (under the homomorphism h which maps the rhs of d to tuples of K) are mapped to LDB instance K with data $h_1(t_1), h_1(t_2), \ldots h_1(t_n)$. Let tuples $h_1(t_1), h_1(t_2), \ldots h_1(t_n)$ in K

have identifiers $ID'_1, ID'_2, \dots ID'_n$. Since h_1 is a l-homomorphism the lineage of tuples $h_1(t_1), h_1(t_2), \dots h_1(t_n)$ points to the same set of base lineage identifiers through Λ_B, i.e.: $\Lambda_B(ID'_1) \cup \Lambda_B(ID'_1), \cup \dots \cup \Lambda_B(ID'_n) = \Lambda_B(ID_1) \cup \Lambda_B(ID_1), \cup \dots \cup \Lambda_B(ID_n) = \Lambda^1_B$.

As a result there exists an ordinary homomorphism $h_1 \circ h$ from the lhs of d, $R_1(\mathbf{x_1}), R_2(\mathbf{x_2}), \dots, R_n(\mathbf{x_n})$ to LDB tuples $h_1(t_1), h_1(t_2), \dots h_1(t_n)$ in K, with identifiers $ID'_1, ID'_2, \dots ID'_n$ and pointing to the same Λ_B with the tuples $t_1, t_2, \dots t_n$ of K_1.

iii) We also have that K l-satisfies d. So from Definition 28 for each ordinary homomorphism h_i that maps the lhd of d to tuples in K pointing to a base lineage identifier set $\Lambda^{d,hi}_B$ we have that there exists a homomorphism h'_i which is an extension of h_i that maps the rhs of d to tuples all with base lineage equal with or contained in Λ^{d,h_i}_B. Homomorphism $h_1 \circ h$ from part (ii) is such a homomorphism because it maps the lhs of d to tuples in K. As we saw in part (ii) it maps the lhs of d to LDB tuples $h_1(t_1), h_1(t_2), \dots h_1(t_n)$ in K and pointing to the same Λ_B with the tuples $t_1, t_2, \dots t_n$ of K_1. As a result each extension $(h_1 \circ h)''$ of $h_1 \circ h$ that maps the lhs of d to tuples $h_1(t_1), h_1(t_2), \dots h_1(t_n)$ in K and it maps the rhs of d to tuples with base lineage equal with $\Lambda^{d,h}_B$ from part (i). If we take the extension $(h_1 \circ h)'$ of $h_1 \circ h$ which is defined through the same extension h' of h that we have in part (i) then $(h_1 \circ h)'$ will map the rhs of d to tuples with data $t'_1, t'_2, \dots t'_n$ in K and all with same base lineage equal with $\Lambda^{d,h}_B$.

From part (i) we have that K_2 includes the tuples $t_1, t_2, \dots t_n$ of K_1 which are mapped to K with the same base lineage through h_1. Instance K_2 also includes the newly "fired" tuples $t'_1, t'_2, \dots t'_n$ with base lineage $\Lambda^{d,h}_B$. Those are mapped from K_2 to the tuples of K which are the images of $(h_1 \circ h)'$ through the extension h' of h of part (i) with base lineage equal with $\Lambda^{d,h}_B$. As a result there exists an l-homomorphism from $K_2 \rightarrow K$ which is the extension of h_1 to the mappings of variables $\mathbf{z}$ defined through the extension of h'. $\square$

Let $< I, J >$ be the result of l-chase of I under a set Σ of tgds. Then if we extract J from $< I, J >$, we have that J is an l-universal solution of the LDB data exchange setting with I as a source LDB and a Σ:

Theorem 18. *If we extract J from the result $< I, J >$ of l-chase of an LDB source instance I under a set of tgds Σ, then J is an l-universal solution.*

Proof. Since Σ_t uses only target relation data, it follows that extracting J from I, J by retaining source IDs as external lineage retains all base lineage information. Let J' be an arbitrary solution. The identity mapping $id : I, \emptyset \rightarrow I, \bar{J}$ is an l-homomorphism. By applying Lemma 2 at each chase step, we obtain an l-homomorphism $h : I, J \rightarrow I, J'$. Hence h is also an l-homomorphism from J to J' (the base IDs that extraction algorithm keeps as external are the same for J and J' since J l-maps to J'). Thus, J maps to every certain-information l-solution. $\square$

Theorem 19. *Consider an LDB data exchange setting with an LDB source instance I with schema S , and T be the target schema and a set of tgds Σ. Let q be a conjunctive query over the target schema T. If J is an l-chase result with inputs I and Σ, then $l - cert(q, I) = R_q(J) \downarrow$ with respect to data and base lineage, where $R_q(J) \downarrow$ is the LDB with the set of all tuples of $R_q(\bar{J})$ that contain only constants along with their base lineage.*

Proof. Let q be a conjunctive query of the form $\exists \mathbf{y}\, \phi(\mathbf{x}, \mathbf{y})$ and let t be a tuple of constants from the source LDB instance I or from rules in Σ. If $t \in l - cert(q, I)$, then $t \in R_q(J)$ with the same base lineage (from Definition 32), since J is a solution. Conversely, assume that $t \in R_q(J) \downarrow$. Then t consists only of constants. Also from LDB query computing Algorithm we have that there exists a homomorphism $g : \phi(\mathbf{x}, \mathbf{y}) \to \mathbf{J}$ such that $g(\mathbf{x}) = \mathbf{t}$. Let J' be an arbitrary solution. Since the result of l-chase J is an l-universal solution (from Theorem 18), there is an l-homomorphism $h_l : J \to J'$. So, considering only data, there exists a homomorphism $h : J \to J'$. Then $h \circ g$ is a homomorphism from $\phi(\mathbf{x}, \mathbf{y})$ to $\bar{J}'$. Homomorphisms are identities on constants, hence $h(g(\mathbf{x})) = \mathbf{h}(\mathbf{t}) = \mathbf{t}$. Now because there exist a homomorphism $g : \phi(\mathbf{x}, \mathbf{y}) \to \mathbf{J}$ and an l-homomorphism $h_l : J \to J'$ we have that all images of $\phi(\mathbf{x}, \mathbf{y})$ under g that exist in J will also exist in J' with same base lineage. Hence t will exist in J' with same base lineage as in J. As a result it will belong to l-cert(q, I). $\square$

From the previous two Theorems, the following result is immediate:

Theorem 20 (Computing LDB certain answers)**.** *Consider an LDB data exchange setting with an LDB source instance I with schema S, T be the target schema and Σ be a set of source to target tgds and weakly acyclic target tgds. Let q be a conjunctive query over the target schema T. We can compute $cert(q, I)$ in time polynomial to the size of LDB I.*

4.3.1 Comparison with Oblivious chase

The *oblivious chase* was defined in [CGK08]:

Definition 39 (Oblivious Chase)**.** The oblivious chase works on an instance K with constants and null values. Let Σ be a set of tgd's and let ξ be a tgd in Σ. Let h be a homomorphism from the lhs of ξ to K such that the image of the lhs of ξ under h does not occur in K. Then an extension h' of h that maps the rhs of ξ to K then an extension h'' of h which maps the existential variables of the rhs of ξ to fresh new null values is applied to the rhs of ξ producing new tuples that are added in K. In that case we say that the oblivious chase "fires" with ξ under h' producing a new instance K', denoted with $K \rightarrow^{\xi,h'} K'$.

The oblivious chase "fires" with h and ξ even if there exists an extension h' that maps the rhs of ξ to K. On the other the same homomorphism h from the lhs of ξ is fired only once with the same tgd ξ. In contrast with our l-chase the oblivious chase does not terminate under arbitrary weakly acyclic tgds. Specifically it terminates only with a set of strongly acyclic tgds which are defined as follows:

Definition 40 (Richly acyclic tgds)**.** The notion of *Richly acyclic tgds* was given in [HS07]: Extend the dependency graph of a tgd by also adding an existential edge from position (R, i) and (S, j) whenever there is a dependency in $\xi in \Sigma$ that has a variable y that appears in position (R, i) (even if y does not appear in the rhs of ξ) of the lhs and an existentially quantified variable z that appears in position (S, j) in the rhs of ξ. The newly created graph is called *extended dependency graph.*

A set of Σ of tgd's is said to be *richly acyclic* if the extended dependency graph of Σ does not have any cycles containing an existential edge.

In our example 18 we have that tgd ξ: $\mathtt{Saw}(\mathtt{witness}, \mathtt{car}), \mathtt{Drives}(\mathtt{p}, \mathtt{car}) \rightarrow \exists D\, \mathtt{Suspects\&Dates}(\mathtt{p}, \mathtt{D})$
is weakly and richly acyclic. In contrast with ordinary chase the oblivious chase will correctly compute not only $Hank, d_1$ but also $Hank, d_4$ as the following Example 24 illustrates. On the other hand if a target tgd ξ' is added, where ξ': $\mathtt{Suspects\&Dates}(\mathtt{p}, \mathtt{d}) \rightarrow \exists D'\, \mathtt{Suspects\&Dates}(\mathtt{p}, \mathtt{D'})$
then ξ, ξ' is a set of weakly acyclic tgds but not richly acyclic. We also show in the Example 24 that indeed the oblivious chase does not terminate:

Example 24. The pseudo-LDB database U_H is shown in Figure 4.16.

We have that there exist four homomorphisms h_1, h_2, h_3, h_4 from the lhs of ξ to U_H. Namely they are:

$h_1: \ witness \rightarrow Cathy, car \rightarrow Honda, p \rightarrow Hank,$
$h_2: \ witness \rightarrow Cathy, car \rightarrow Mazda, p \rightarrow Jimmy,$
$h_3: \ witness \rightarrow Amy, car \rightarrow Toyota, p \rightarrow Billy,$
$h_4: \ witness \rightarrow Amy, car \rightarrow Honda, p \rightarrow Hank$

So the oblivious chase will fire with ξ and the extensions h'_1, h'_2, h'_3, h'_4 that map variable *date* to d_1, d_2, d_3 and d_4 respectively, producing the instance

ID	**Saw**(witness,car)
11,1	Cathy,Honda
11,2	Cathy,Mazda
12,1	Amy,Toyota
12,2	Amy,Honda

ID	**Drives**(name,car)
21,1	Hank,Honda
22,1	Jimmy,Mazda
23,1	Billy,Toyota

Figure 4.16: D_1: Pseudo-LDB horizontal U_H of ULDB U.

ID	**Suspects&Dates** (suspect,date)
31	Hank,d_1 ?
32	Jimmy,d_2 ?
33	Billy,d_3 ?
34	Hank,d_4 ?

Figure 4.17: Target relation *Suspects&Dates* of U_H^1

U_H^1 which has the relation in U_H along with the *Suspects&Dates* relation of Figure 4.17

Then the oblivious chase will terminate because there exists no new homomorphism under which the lhs of ξ is mapped to U_H^1.

On the other hand of we add target tgd ξ': $\texttt{Suspects\&Dates(p, d)} \rightarrow \exists D' \, \texttt{Suspects\&Dates(p, D')}$
then the oblivious chase will fire with the following four homomorphisms from the lhd of ξ:
$h_{21}: \ p \rightarrow Hank, d \rightarrow d_1,$
$h_{22}: \ p \rightarrow Jimmy, d \rightarrow d_2,$
$h_{23}: \ p \rightarrow Billy, d \rightarrow d_3,$
$h_{24}: \ p \rightarrow Hank, d \rightarrow d_4$
with the extensions $h'_{21}, h'_{22}, h'_{23}, h'_{24}$ that map variable *date* to d_5, d_6, d_7 and d_8 respectively. But then the homomorphism $h''_{21}: \ p \rightarrow Hank, d \rightarrow d_5$ is a new homomorphism under which ξ_2 has not fired.

4.4 Data exchange for Uncertain Databases with Lineage (ULDBs)

For ordinary databases we had the following definition for certain answers: given a query q posed over a certain instance I with null values, then: $certain(q, I) = \{ \cap(J) \mid \text{where } J \text{ is a solution } \}$. However, for a ULDB data exchange setting, if we take the intersection of the answers to the query over all possible instances, it is easy to see that we will mostly derive an empty set of answers. Moreover, this does not agree with the intuition that only one of the possible instances is the correct one. If so, then taking the intersection between facts appearing in the one "true" instance and all other "false" possible instances is meaningless. Thus, in order to semantically define certain answers in a ULDB data exchange setting we think as follows (similar intuition was applied in the definition of certain answers in [ALP08b]): Each possible instance D_i of a ULDB source instance I is an LDB and gives rise to an LDB data exchange problem with D_i as its source instance and the same set of dependencies Σ. Since we do not know which possible instance is the one that captures the "truth" it is natural to consider them all but "separately". We note that since possible instances are LDBs, they are allowed in them duplicates of data if they have different lineage. We retain this property, i.e. we do not perform any kind of duplicate elimination.

As a result when we pose a query to any of the solutions of all LDB data exchange problems that arise from the D_i possible instances of a source ULDB we expect to get the corresponding certain answer. We thus have the following definition:

Definition 41 (ULDB Certain answers)**.** Consider a data exchange setting $(\mathbf{S}, \mathbf{T}, \mathbf{\Sigma} = \mathbf{\Sigma_{st}} \cup \mathbf{\Sigma_{t}})$. If I is a source ULDB instance with $PI(I) = D_1, \ldots, D_n$ and q a query over target schema $\mathbf{T}$ then the certain answers of q with respect to I, denoted as *uldb-cert*(q, I) is a complete ULDB C (with no nulls) with $PI(C) = D''_1, \ldots, D''_n$ such that:
each D''_i is an l-certain answer of the LDB data exchange problem with D_i as its LDB source instance and Σ as its set of constraints.

Figure 4.18 illustrates the notion of ULDB certain answers. We want to give an appropriate procedure to compute *uldb-cert*(q, I) which will be based on l-chase for databases with lineage. We do not to explicitly produce the possible instances of the source instance, which is computationally expensive, but rather compute in polynomial time directly a ULDB that will be used for the computation of *uldb-cert*(q, I). On the other hand the semantics of the result of this procedure should be the following: the possible instances of the

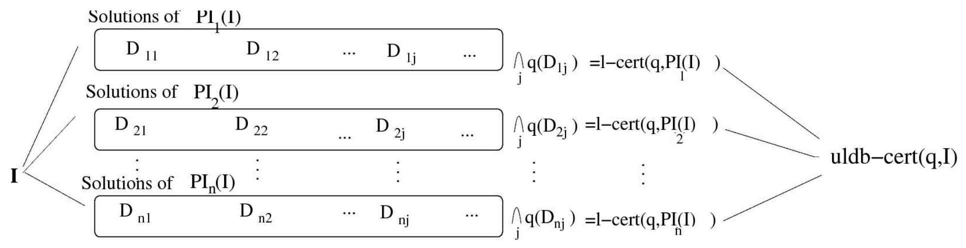

Solutions of $PI_1(I)$
D_{11} D_{12} ... D_{1j} ...
$\bigcap_j q(D_{1j}) = l\text{-}cert(q, PI_1(I))$
Solutions of $PI_2(I)$
D_{21} D_{22} ... D_{2j} ...
$\bigcap_j q(D_{2j}) = l\text{-}cert(q, PI_2(I))$
I
Solutions of $PI_n(I)$
D_{n1} D_{n2} ... D_{nj} ...
$\bigcap_j q(D_{nj}) = l\text{-}cert(q, PI_n(I))$
uldb−cert(q,I)

Figure 4.18: ULDB Certain Answers

ULDB that will be the result of our uncertain-chase (which we will denote by u-chase) procedure should be the same with the possible instances that we would have if we first took the possible instances of the ULDB instance and apply l-chase to each one of them.

We will make use of a way to transform a ULDB into a "pseudo-LDB" called its "Horizontal Relation" that we also presented on Chapter 2. A similar approach was used in an algorithm in [BSH+08a] to correctly compute answers of CQs over ULDBs in PTIME without producing all possible instances. Horizontal relations were first defined in [SUW09] in a variation without lineage. Intuitively in order to take the Horizontal database of a ULDB we "flatten" each alternative so that it will become a new x-tuple (with no other alternatives), but retain the lineage information (now referring to tuples and not to alternatives). As is stated in [BSH+08a], we have that since the size of D_H is the same as the size of ULDB relations R_1, $R_2, \dots, R_n$, complexity does not increase when we compute a horizontal relation of a ULDB. We repeat the definition here:

Definition 42 (Horizontal Relation, Horizontal Database)**.** Let R be an relation of a ULDB. We define the *Horizontal relation* of R and denote it with R_H the LDB (with no uncertainty -no alternatives- but with lineage): $R_H = \{$ tuples $s(i,j) \mid s(i,j)$ is an alternative in $R\}$.

Let D be a ULDB with x-relations $\{R_1, \dots, R_n\}$. We define the Horizontal database of D and denote it with D_H the LDB: $D_H = R_{1H}, \dots, R_{nH}$ such that $\forall k$, $k \in [1,n]$: $R_{kH} = \{$ tuples $s(i,j) \mid s(i,j)$ is an alternative in $R_k\}$

Definition 43 (U-Chase sequence)**. input:** A source schema **S**, a target schema **T**, a source ULDB D with x-relations $\{R_1, \dots, R_n\}$ and Σ a set of source-to-target and target tgds. **output:** a ULDB D'. Steps:
1. From ULDB source D create the horizontal database D_H. 2. Since D_H is an LDB we can now apply an l-chase sequence with D_H as the source LDB instance and constraints Σ and produce D'_H. 3. Create a ULDB D' that is the union of i) the source ULDB relations $\{R_1, \dots, R_n\}$ and of ii) target ULDB relations that are created from the target horizontal relations of D'_H by treating each LDB tuple as a maybe x-tuple with one alternative (so with symbol '?') and retaining the lineage relationship of D'_H. 4. Return D'.
A *terminating u-chase* sequence is the one that applies a terminating l-chase sequence.

Note that in the u-chase result we do not regroup alternatives of target relations to x-tuples, like ULDB conjunctive query algorithm in [BSH+08a] does. The possible instances of our ULDB output will be the same even without regrouping due to the constraints posed by lineage information. In addition when we compute certain answers of a CQ (which is the reason we defined u-chase for), ULDB conjunctive query algorithm will then regroup alternatives to x-tuples.

4.4.1 U-chase result for our example

Example 25. Let us consider the ULDB data exchange problem of our running Example 18: The source ULDB I is shown in Figure 4.1. We follow the steps of u-chase: We first create the horizontal relations of the source database **A**. Since horizontal relations are LDBs we apply l-chase to them and get an intermediate LDB Suspects&Dates$_H$. In our example, this is the result of a single l-chase step since our tgd will not l-fire again.

We form a ULDB from the LDB result of l-chase: Each LDB horizontal tuple becomes a ULDB x-tuple with one alternative and we retain all lineage connections, now referring to alternatives. We have to add '?' at each x-tuple because we no longer have an LDB: In LDBs all tuples are certain (since LDBs have one possible instance). Some of those '?' symbols might be extraneous, but we can remove them in polynomial time, as it proven in [BSH+08a]. The extracted Suspects&Dates target relation of U-chase result is the same as the one we expected in Example 18, shown in Figure 4.2. Let us denote with J' this materialized target instance of relation Suspects&Dates which is the target result of U-chase. This result can be used in order to compute ULDB certain answers (*uldb-cert*(Q, I)): $Q(J'_{\downarrow})$ are the extracted certain answers of query Q of Example 18. Indeed the result of $Q(J'_{\downarrow})$ is the same with the ULDB certain answers of Q which we intuitively expected, shown in Figure 4.3. So for each possible instance of the source database **A** we get the l-certain answer that would be real if we had a data exchange problem with that instance. The retained lineage will logically restrict the possible instances of the target ULDB. For example tuples having $11, 1$ in their base lineage will never coexist in a possible instance with tuples having $11, 2$ in their lineage, even though the "data part" of source x-tuple 11 would have been removed. This is one of the aims of data exchange: to be able to compute certain answers only from target database when the source data in no longer available. We are now going to prove that the result of u-chase result can always be used in order to compute ULDB certain answers.

4.4.2 Complexity of Computing ULDB certain answers

First we prove that any u-chase sequence on a well-behaved ULDB U that creates a ULDB U' is equivalent to applying an l-chase sequence to each of the possible instances of U. Then in Lemma 3 we prove that a terminating u-chase on U produces a ULDB with possible instances which are the result of a terminating l-chase applied on each of the possible instances of U. Finally we have the last Theorem 22 which states that for a well-behaved source ULDB instance and a set of weakly acyclic tgds we can use the result of u-chase in order to compute ULDB certain answers of conjunctive queries in polynomial time. Its proof will make use of Theorem 21 and Lemma 3:

Theorem 21. *Suppose we start with a well-behaved ULDB D_0 and after a u-chase sequence steps we arrive at a ULDB D_j. Suppose that the possible instances of D_0 are: $D_0^1, D_0^2, \ldots, D_0^i$. Suppose that the possible instances of D_j are: $D_j^1, D_j^2, \ldots, D_j^{i'}$. Then the following holds: i) $i = i'$ and ii) every D_j^k comes from a $D_0^{k'}$ after a sequence of l-chase steps.*

Proof. *i*) Since we consider only tgds at each u-chase step only one alternative can be generated that points to previous alternatives with well-behaved lineage. So from the previous lemma we have that the result of a u-chase preserves the instances of the ULDB it started with. As a result the number of the possible instances of the result of a u-chase sequence will be the same as the number of possible instances of initial source D_0.

ii) By induction on the number of l-chase steps j.

Base step: The base step is trivial for $j = 0$ since if we start with D_0 and perform no u-chase then the result will be again D_0 with the same possible instances.

Inductive hypothesis: Suppose that if we start with a ULDB D_0 with possible instances: $D_0^1, D_0^2, \ldots, D_0^i$ and perform a sequence of $j < n$ u-chase steps and arrive at a ULDB D_j with possible instances: $D_j^1, D_j^2, \ldots, D_j^i$, then every D_j^k comes from a $D_0^{k'}$ after a sequence of l-chase steps.

As a result for $j = n-1$ ULDB D_{n-1} has possible instances: $D_{n-1}^1, D_{n-1}^2, \ldots, D_{n-1}^i$.

We prove that then if we start with ULDB D_{n-1} with possible instances: $D_{n-1}^1, D_{n-1}^2, \ldots, D_{n-1}^i$ and perform a u-chase step we will arrive at a ULDB D_n with possible instances: $D_n^1, D_n^2, \ldots, D_n^i$ such that then every D_n^k comes from a $D_{n-1}^{k'}$ after an l-chase step or is equal with a $D_{n-1}^{k'}$.

Let the n^{th} u-chase step use tgd d. Let d be a tgd of the form:
$\forall \mathbf{x}\big(R_1(\mathbf{x_1}), R_2(\mathbf{x_2}), \ldots, R_n(\mathbf{x_n})\big) \rightarrow \exists \mathbf{y}\big(T_1(\mathbf{x'_1}, \mathbf{y_1}), T_2(\mathbf{x'_2}, \mathbf{y_2}), \ldots, T_m\ (\mathbf{x'_m}, \mathbf{y_m})\big)$.

Then u-chase will first create the horizontal relations of all R_i, so alternatives $s(i, j)$ of ULDB D_{n-1} will become identifiers of LDB tuples.

Since u-chase "fires" it generated alternatives $T_1(t_1), T_2(t_2), \ldots, T_n\ (t_n)$ each with same lineage, i.e.:

$\forall i \in [1, \ldots, n]: \quad \lambda(t_i) = \{s(i_{r_1}, j_{r_1}), \ldots, s(i_{r_n}, j_{r_n})\}$

As a result all $T_1(t_1), T_2(t_2), \ldots, T_n(t_n)$ will belong to all the possible instances of D_n that have all $s(i_{r_1}, j_{r_1})$, $s(i_{r_2}, j_{r_2}), \ldots, s(i_{r_n}, j_{r_n})$ in them. The notation represents the fact that $s(i_{r_1}, j_{r_1})$ is the alternative where $R_1(\mathbf{x_1})$ is mapped to in current u-chase step.

Let $\Lambda_B^1 = \lambda_B(s(i_{r_1}, j_{r_1})) \cup \ldots \cup \lambda_B(s(i_{r_1}, j_{r_1}))$.

If Λ_B^1 contains two base tuples of the form $s(i, j)$ and $s(i, j')$ with $j \neq j'$ then none of the newly generated tuples appear in any possible instance of D_n. This holds because $s(i, j)$ and $s(i, j')$ correspond to two alternative values of a same base x-tuple. But alternative values of one x-tuple are mutually exclusive and can never coexist in a possible instance. In general such alternatives that can never exist in any possible instance are called "extraneous" [BSH+08a].

Thus none of $T_1(t_1), T_2(t_2), \ldots, T_n(t_n)$ belong to any of the possible instances of D_n. Hence its possible instances remain the same as the ones of D_{n-1}. From the inductive hypothesis ULDB D_{n-1} has possible instances: $D_{n-1}^1, D_{n-1}^2, \ldots, D_{n-1}^i$ and every D_{n-1}^k comes from a $D_0^{k'}$ after a sequence of l-chase steps.

If Λ_B^1 does not indicate "extraneous" alternatives. Then pick an arbitrary D_{n-1}^k that is a possible instance of D_{n-1}.

If even one of alternatives of Λ_B^1 does not occur in D_{n-1}^k then none of $T_1(t_1), T_2(t_2), \ldots, T_n(t_n)$ will belong to it. This holds because for well-behaved lineage an alternative exists in a possible instance if and only if all of the alternatives in its lineage exist in it (see Definitions 6, 7). The existence of an alternative in a possible instance is in turn determined (in well-behaved ULDBs) from the choices of base x-tuples (Theorem 1). So in this case the 'firing' of u-chase does not alter possible instance D_{n-1}^k.

Else we must have that all alternatives of Λ_B^1 occur in D_{n-1}^k. Then all $T_1(t_1), T_2(t_2), \ldots, T_n(t_n)$ will belong to the corresponding possible instance D_n^k of ULDB D_n that is the result of u-chase step. But since u-chase "fires" we have that there exists a homomorphism h from the lhs of d to horizontal relations of $R_1, \ldots, R_n$ (so from R_i alternatives of ULDB D_{n-1} according to u-chase Algorithm 43) such that there exists no homomorphism h' extension of h that maps the rhs of d to tuples of the horizontal relations of $T_1, \ldots, T_m$ (to T_i alternatives of ULDB D_{n-1}) so that the base lineage of each one is equal with Λ_B^1 (from Definition 35 of l-chase step). But since all alternatives of Λ_B^1 occur in D_{n-1}^k we will have l-chase of in D_{n-1}^k will also "fire" and produce exactly $T_1(t_1), T_2(t_2), \ldots, T_n(t_n)$ with the same lineage. Because if l-chase did not "fire" in D_{n-1}^k this would mean that there existed $T_1(t_1), T_2(t_2), \ldots, T_n(t_n)$ in D_{n-1}^k all of them with lineage Λ_B^1. But then those $T_1(t_1), T_2(t_2), \ldots, T_n(t_n)$ would also exist in the horizontal relation of $T_1(t_1), T_2(t_2), \ldots, T_n(t_n)$ with that lineage, causing u-chase not to fire. □

Lemma 3. *Suppose that we have a data exchange problem with a well-behaved ULDB source D_0 with $PI(D_0) = D_0^1, D_0^2, \ldots, D_0^n$ and a set Σ of tgds. Then the result of a terminating u-chase with Σ is a ULDB D_j with $PI(D_j) = D_j^1, D_j^2, \ldots, D_j^n$ such that for every $i = 1 \ldots n$ we have that D_j^i is an l-universal solution (if we extract target relations) of the LDB data exchange problem with D_0^i as the source and Σ as constraints.*

Proof. Suppose that u-chase terminates. Then u-chase does not fire with any of the tgds in Σ. We will suppose that l-chase would fire on one of D_j^i with a tgd d in Σ and yield a contradiction.

Let d be a tgd in Σ with the following form:
$\forall \mathbf{x}\big(R_1(\mathbf{x_1}), R_2(\mathbf{x_2}), \ldots, R_n(\mathbf{x_n})\big) \rightarrow$
$\exists \mathbf{y}\big(T_1(\mathbf{x'_1}, \mathbf{y_1}), T_2(\mathbf{x'_2}, \mathbf{y_2}), \ldots, T_m(\mathbf{x'_m}, \mathbf{y_m})\big)$.

Let D_j^k be a possible instance of D_j. If l-chase fires on D_j^k under d then there exists a homomorphism h from the lhs of d to tuples $R_1(t_1), \ldots, R_n(t_n)$ of D_t^n with IDs $ID(t_1), ID(t_2), \ldots, ID(t_n)$ such that there exists no homomorphism h', extension of h, that maps the rhs of d to tuples $T_1(t'_1), \ldots, T_m(t'_m)$ of D_t^n so that the base lineage of each one of $T_1(t'_1), \ldots, T_m(t'_m)$ is equal with $\Lambda_B^1 = \lambda_B(ID(t_1)) \cup \ldots \cup \lambda_B(ID(t_n))$ of D_t^n. Then u-chase would also fire, which is a contradiction, because:
The only case for u-chase to not fire is if tuples $T_1(t'_1), \ldots, T_m(t'_m)$ with base lineage for each one of them be Λ_B^1 to exist in the horizontal relations of T_i. But if $T_1(t'_1), \ldots, T_m(t'_m)$ with base lineage for each one of them be Λ_B^1 already existed in the horizontal relation ot T_i then they would all exist in the same possible instances, because they have the same base lineage. Moreover they will exist in D_j^k because this possible instance contains all base tuples of their lineage. So then l-chase would not fire.

Since l-chase does not fire with any of the tgds in Σ and for any of D_t^i, we have from Theorem 18 that the result of a terminating l-chase is an LDB solution. So for every $i = 1 \ldots n$ we have that D_j^i is an l-universal solution (if we extract target relations) of the LDB data exchange problem with D_0^i as the source and Σ as constraints. □

From Lemma 3, Theorem 21 and from Theorems 20, of the previous section we have the following:

Theorem 22 (ULDB Certain Answers)**.** *Consider a ULDB data exchange setting with a well-behaved ULDB source instance I with schema S, T be the target schema and Σ be a set of source to target tgds and weakly acyclic target tgds. Let q be a conjunctive query over the target schema T. We can* *compute uldb-cert(q, I) in time polynomial to the size of I using the result of u-chase.*

4.4.3 Complexity of a ULDB data exchange problem that includes egds in its set of constraints Σ

Let us modify a little our running example by adding one new attribute `signa- ture` to relation `Saw` which contains the name that appears in the signature of a witness. Suppose that *Cathy* is not certain if the car she saw was a $Honda$ or a $Mazda$ but we always have her name in the signature. On the other hand *Amy* saw a $Toyota$ car but we have uncertainty if the witness report has a signature with name *Amy* or *Annie* (for example due to a not clear signature). Now we suppose that the private investigator also stores information about the name of the witness and her signature. Hence target `Suspects&Dates` now has schema: {`witness,signature,suspect,date`}. So we change the source to target tgd to the following ξ_1: `Saw(witness, signature,car), Drives(p,car)` $\rightarrow \exists D$ `Suspects&Dates(witness, signature,p,D)`. Now let us consider that we also have the target egd ξ_2: `Suspects&Dates( witness,signature,suspect,date)` $\rightarrow$ `(witness = signature)`.

Since we do not know which instance captures the "truth" we would want all the possible instances to satisfy the egd. Intuitively the ULDB result of u-chase with ξ_2 should represent only the two possible instances that do not contain any result derived from the possible instances that contain in `Saw` a tuple with name *Amy* and signature *Annie*. The reason is that any possible instance of the source containing tuple $Saw(Amy, Annie, Toyota)$ will give a failing l-chase. But we prove in the following theorem that even asking for a solution in a ULDB data exchange problem that contains egds in its constraints is an NP-hard problem. The proof will be a reduction from 3-coloring. A UDLB solution will be a target instance that satisfies the constraints. Satisfaction for ULDBs of a constraint d (egd or tgd) is formally defined in Definition 45 which uses the following Definition 44 for egds and LDBs. Finally we note that for an LDB data exchange setting with weakly acyclic tgds, we can include egds and still compute l-certain answers in polynomial time.

Definition 44 (LDB satisfaction of an egd)**.** Let D be an LDB and d be an egd of the form: $\forall \mathbf{x}\big(R_1(\mathbf{x_1}), R_2(\mathbf{x_2}), \dots, R_n(\mathbf{x_n}) \rightarrow (x_1 = x_2)\big)$. We will say that LDB D l-satisfies d, if for each homomorphism h that maps $R_1(\mathbf{x})$, $R_2(\mathbf{x}), \dots, R_n(\mathbf{x})$ to tuples: $R_1(t_1)$ with $ID = ID_1$, $R_2(t_2)$ with $ID = ID_2$, $\dots$, $R_n(t_n)$ with $ID = ID_N$ in D, then $h(x_1) = h(x_2)$.

Definition 45 (ULDB satisfaction of an egd/tgd)**.** Let D be a ULDB instance. Suppose that the possible instances of D are LDBs: $D^1, D^2, \ldots, D^n$. Let d be a tgd or an egd constraint. Then D will ULDB satisfy (u-satisfy) d if for every i the following holds: D^i l-satisfies d.

Theorem 23. *Consider a UDLB data exchange setting with a well-behaved ULDB source I and a set of dependencies Σ that can contain egds as well as tgds and a CQ Q. Then the following problems are an NP-hard: i) "Is there a solution of the data exchange setting I and Σ?", ii)"Is a tuple t in uldb-cert(Q, I)?".*

Proof. (sketch) i) By reduction from 3-coloring: Let G be a graph. Consider a ULDB relation `Color(vertex,coloring)` that for each vertex v_i of G has one x-tuple of the form: $Color(v_i, blue)\,||\,Color(v_i, red)\,||\,Color(v_i, green)$. The ULDB source (with empty lineage) I has relation `Color` and one more relation `edge(x,y)`. Relation `edge` contains an x-tuple with one alternative (v_k, v_l) if there exists an edge in G that connects vertices v_k and v_l and an x-tuple with one alternative (v_i, v_i) for every vertex v_i of G. Consider the following 2 source-to-target (copy) tgds: $Color(x, y) \to Color_2(x, y)$ and $edge(x, y) \to e_2(x, y)$ and the following 3 target egds: $e_2(x, y), Color_2(x, blue)$, $Color_2(y, blue) \to x = y$,
$e_2(x, y), Color_2(x, red), Color_2(y, red) \to x = y$ and
$e_2(x, y), Color_2(x, green), Color_2(y, green) \to x = y$. Then it is easy to see that there exists a solution of this ULDB data exchange problem if and only if there exists a 3-coloring for the graph that is represented by the source relations.
ii) Consider the data exchange problem in the proof of part (i). Let $Q(x, y)$:- $e_2(x, y)$ and let $t \in Q(x, y)$ be an arbitrary arc of G. Then due to the data exchange constraints $t \in$ *uldb-cert*(Q, I) if and only if t is an arc of a graph G that has a 3-coloring. $\square$

Chapter 5

Efficient Query Computing for Uncertain Possibilistic Databases with Lineage

We propose an extension of possibilistic databases that also includes lineage (aka provenance). The introduction of provenance makes our model closed under selection with equalities, projection and join. In addition the computation of query computing with possibilities is *polynomial*, in contrast with current models that combine provenance with probabilities and have $\#P$ complexity.

5.1 Introduction

Modeling, representing and manipulating uncertain data has gained a lot of research attention. There has been a plethora of models that capture different kinds of uncertainty: i) when an attribute can take a value from a finite set of *alternatives* (model of or-sets), ii) when the existence of a whole tuple is not certain (model of ?-tuples). The last two kinds can be combined yielding the *x-tuple* and *x-relation* model $R^a_?$ [BSH+08a, GT06]. On top of uncertain models we can also put "confidence" values. For example in the model of ?-tuples if we attach on each tuple the probability of the event that this tuple is indeed present in our data we yield the model of probabilistic databases [DS04].

One of the key aspects of an uncertain database framework is how efficient it can compute queries. Consider a query Q and an uncertain database U. An uncertain database U represents a set of *possible worlds* $PW(U)$. One naive approach in order to compute Q over U would be to compute first all the possible worlds of $PW(U)$ and pose the query over each one of them. This approach is not efficient since computing the possible worlds of an uncertain database can be intractable in the size of the data. For example if we have an uncertain relation with n x-tuples and each one of which has m different alternatives then the possible worlds are m^n. In contrast we would like to be able to efficiently compute query Q posed directly on uncertain database U and the result $Q(U)$ to be a new uncertain database which can be represented in our model with the correct semantics, i.e., we have that $PW(Q(U)) = Q(PW(U))$ [IL84]. If this holds for an uncertainty model and a query language L we say that this model is *closed* under L.

Possibilistic databases extend the x-tuple model by attaching on each alternative value degrees from a possibility distribution. In [BP05] it was shown that possibilistic databases are not closed under: i) selection with a condition that involves different attributes, ii) projection that performs duplicate elimination in the tuples of the answer and iii) under the join operator. In addition existing database models with provenance that attach "belief values" by using probabilities have high complexity $\#P$ [DS04, GT06, RPT11, STW08]. We solve the first problem and for the second problem we offer a suitable alternative to probabilities by proposing a new model which extends possibilistic databases by adding provenance. The proposed model has the following benefits:

- *Closed* under: i) selection involving equalities even over different attributes, ii) projection even after duplicate elimination and iii) join. This property is a result of the introduction of provenance in the possibilistic model.
- The possibility values of each tuple alternative in the answer of a query involving the above three operators are computed in polynomial time.

Our main contribution is that we define operators for equality select, projection (wth duplicate elimination) and join that can be posed directly on a database expressed in our model of provenance and possibilities without the need to compute first all the possible worlds. The result of each operator is a new database of our model that has the correct semantics: its possible worlds are the same with the ones we would have if we first computed all the possible worlds and pose queries over them. Moreover our operators compute data and possibilities for each alternative of the result in polynomial time.

We think that the employment of possibilities instead of probabilities in our model offers more suitable modeling of alternative belief values, due to the qualitative nature of possibilities. For example suppose that we want to model the fact that a witness *Amy* is uncertain of whether she saw a *Mazda* or a *Toyota* car but she believes that more likely it was a *Mazda*. These kinds of real-life situations are well-represented though possibilistic theory. In addition even when only probabilities of alternatives are available, there exists a way to "translate" probability values to possibilities such that the more probable events will also be more possible, as it is intuitively expected [DFMP04].

5.1.1 Related Work

The possibilistic model is not closed for SPJ queries because it is not powerful enough to pose logical constraints on the alternative values that tuples can take of the answer of a query (e.g., indicate that two alternatives of two different tuples cannot coexist in a possible world [BP05]). Recent work [BPP09] efficiently computes SPJ queries over a limited possibilistic model (specifically where only one alternative has possibility 1 and all others have $1 - a$) and the answers of the queries include only tuples appearing in a complete possible world (a world with possibility 1). In contrast our approach returns all tuples appearing in any possible world and does not require the initial data to have this limitation in its possibilities.

Many models have been proposed that are able to handle uncertainty and keep track of the provenance of data which is usually modeled though semiring annotations on data [BT07, GKT07, KIT10]. Those models are closed under positive relational algebra but if probabilistic confidence values are added on each possible alternative tuple then the computation of the probabilities of the answer of a query that involves projection (with duplicate elimination) is intractable (specificaly #P) [DS04, GT06, RPT11, STW08]. The provenance used in Trio system [BSH$^+$08a] is one out of many kinds of provenance that semirings can model [GKT07]. Our proposed model extends possibilistic databases by adding Trio's provenance used in the Uncertain and Lineage ULDB model [BSH$^+$08a] (where provenance is called "lineage"). The reason we choose this model is because it expresses tuple uncertainty and provenance tracking over the $R^a_?$ (or-set and ?-tuple) x-tuple model which is also used in possibilistic databases[1].

5.2 Properties of the Proposed Model

In this section we illustrate the key aspects of possibilistic databases and of the provenance/lineage semiring of Trio. We also investigate how we can

[1]Note that our proposed model that extends possibilistic databases with Trio's provenance can be equivalently regarded as an extension of the ULDB model with possibilistic confidence values on each alternative

combine uncertainty (x-tuples), provenance (Trio's lineage semiring) and possibilities. We begin with stating the basic properties of the Possibility Theory [BPP09].

5.2.1 Possibility Theory

A possibility distribution is a function π from a domain X to the interval $[0, 1]$. Possibility $\pi(a)$ is a qualitative measure expressing the degree of "how possible" it is for the considered variable to take the value a. Each possibility distribution has a normalization condition posing the constraint that at least one of the values of X is completely possible, i.e., has possibility 1. We use a discrete domain of possible values and we denote with $\{a_1{:}\pi_1, \ldots, a_n{:}\pi_n\}$ the fact that for each $i = 1 \ldots n$ value a_i has possibility π_i. The axioms of possibility are the following: i) $\Pi(X) = 1$, ii) $\Pi(\emptyset) = 0$, iii) $\Pi(E_1 \cup E_2) = \max(\Pi(E_1), \Pi(E_2))$, iv) $\Pi(E_1 \cap E_2) \leq \min(\Pi(E_1), \Pi(E_2))$ and when E_1 and E_2 are not-interactive: $\Pi(E_1 \cap E_2) = \min(\Pi(E_1), \Pi(E_2))$. For the events E and $\bar{E}$ (opposite of E) the only valid relation is: $\max\{\Pi(E),\ \Pi(\bar{E})\} = 1$. Apart from possibility each event has a necessity measure N which is dual with Π and their relation is expressed through: $N(E) = 1 - \Pi(\bar{E})$.

5.2.2 The Proposed Model: Combining Uncertainty, Possibilities and Provenance

Possibility theory can be naturally adapted to the model of x-relations and the semiring of Trio [BSH+08a]. In x-relations model we no longer have ordinary tuples. Instead we have x-tuples which include a bag of possible ordinary tuples, called *alternatives*. The semantics are the following: on each possible world at most one of the alternatives of an x-tuple can be true. If from an x-tuple we can select none of its alternatives then this is a maybe-xtuple annotated with symbol '?'. It is then straightforward that we can combine possibilistic theory and x-tuples in the following way: Suppose that we have an uncertain database which contains x-tuples. We attach on each alternative a possibility degree and on each x-tuple at least one alternative should be assigned with possibility 1 (the most possible one(s)). Furthermore for each x-tuple with a '?' symbol we attach to it a necessity degree less than 1 and to all other x-tuples necessity equal to 1 (since an alternative of each one of them is always possible). We do not have to explicitly attach a possibility degree on each x-tuple since it is equal to the minimum possibility of each alternative. So we always begin with an uncertain database containing x-tuples with possibility degrees on each alternative and necessity degrees on each x-tuple.

Trio's provenance (called "lineage" in Trio) semiring works as follows: If we pose queries over initial data we want to keep track of the provenance of the answers, i.e., from which data the answers are derived from. In order to do this efficiently we attach a *unique* identifier i over each x-tuple. We also identify the alternatives of each x-tuple: In general the pair (i, j) identifies the j-th alternative of x-tuple i. If an alternative with data t is a result from two other alternatives t_1 and t_2 but can also be the result of our query combining two other alternatives t_3 and t_4 then we have for its lineage: $\lambda(t) = (id(t_1) \wedge id(t_2)) \vee (id(t_3) \wedge id(t_4))$. We note that lineage plays a double role: it relates answers of queries to the data they are coming from and also poses logical restrictions: an alternative can be true only in a possible world in which its lineage is true. We note that initial data have empty lineage. Initial data with empty lineage is defined as *base* data. We refer to [BSH+08a] for more details about lineage and possible worlds. We borrow from the same work the general setting of our following running example.

5.2.3 Running Example

Consider x-relation $Saw(witness, car)$ having two x-tuples with two alternatives each. Suppose that witness Amy saw a car near a crime-scene but she was not sure if it was a $Mazda$ or a $Toyota$ car. Moreover she believed it was more possible that the car was a $Mazda$ and a little less possible that it was a $Toyota$. The first tuple has identifier 11 and the second 12. We separate different alternatives of a same x-tuple with || symbol. After each alternative we attach its possibility and after each x-tuple its necessity measure, i.e., $<t'_1{:}a||t'_2{:}\text{b}>{:}c$ is an x-tuple with necessity c that has two alternatives: alternative with data t'_1 has possibility a and alternative t'_2 has possibility b. Suppose also that in x-relation $Drives(person, car)$ we encode uncertainty about who is driving a car of a specific brand. The uncertain database U with Trio's provenance of our running example is:

Saw(witness,car)=
{11<Amy, Mazda:1||Amy, Toyota:0.8>:1,
12<Billy, Mazda:0.4 || Billy, Lexus:1>:1}
Drives(person,car)=
{21<Hank, Mazda:0.6||Hank,Toyota, :1>:1}

There is a total of $2^3 = 8$ possible worlds. Suppose that we pose query Q_2 which is a projection of attribute $person$ on the result of query Q_1 which is the join of Saw and $Drives$ over common attribute car, i.e.: $Q_1 = Saw \bowtie_{car=car} Drives$ and $Q_2 = \pi_{person}(Q_1(U))$. Only five of the possible worlds include answers over Q_1 (and Q_2). Those worlds are:

W1: Saw={11,1<Amy, Mazda:1>:0.2,
12,1<Billy, Mazda:0.4>:0}
Drives={21,1<Hank, Mazda:0.6>:0}
$\Pi(W1) = 0.4$, $N(W1) = 0$
W2: Saw={11,1<Amy, Mazda:1>:0.2,
12,2<Billy, Lexus:1>:0.6}
Drives={21,1<Hank, Mazda:0.6>:0}
$\Pi(W2) = 0.6$, $N(W2) = 0$
W3: Saw={11,2<Amy, Toyota:0.8>:0,
12,1<Billy, Mazda:0.4>:0}
Drives={21,1<Hank, Mazda:0.6>:0}
$\Pi(W3) = 0.4$, $N(W3) = 0$
W4: Saw={11,2<Amy, Toyota:0.8>:0,
12,1<Billy, Mazda:0.4>:0}
Drives={21,2<Hank, Toyota:1>:0.4}
$\Pi(W4) = 0.4$, $N(W4) = 0$
W5: Saw={11,2<Amy, Toyota:0.8>:0,
12,2<Billy, Lexus:1>:0.6}
Drives={21,2<Hank, Toyota:1>:0.4}
$\Pi(W5) = 0.8$, $N(W5) = 0$
For example the possibility of PW_1 is equal to the minimum of the possibilities of its alternatives, so with $\min\{1,\ 0.4,\ 0, 6\} = 0.4$. Its necessity is equal to 1 minus the maximum possibility from the possibilities of alternatives which do *not* belong to this world have: $1 - \max\{0.8,\ 1,\ 1\} = 1 - 1 = 0$. The necessity of x-tuple $(11, 1)$ is equal to 1 minus the maximum possibility of the other alternatives (in our case only alternative $11, 2)$ of initial x-tuple 11, i.e., equal to $1 - \max\{0.8\} = 0.2$.

For the answers of queries we have similar semantics with the ones defined for probabilities in [DS04]: The answer of a query Q is a set of alternatives and their possibilities. Intuitively for the answer of Q_1 we should have:
$\mathbf{Q_1(U)}$={31<Amy, Mazda, Hank:0.6>:0,
32<Billy, Mazda,Hank:0.4>:0
33<Amy, Toyota,Hank:0.8>:0}
For example alternative $(Amy, Mazda, Hank)$ appears in W_1, a world with possibility 0.4 and in W_2 with possibility 0.6 (while both necessities are 0 - note that only a world whose all tuples have possibility 1 has necessity greater than 0). As a result in the answer of query Q_1 we want to have a tuple with data $(Amy, Mazda, Hank)$ with possibility the union of the events that this tuple appears in W_1 or in W_2. So with the maximum of the possibilities of 0.4 and 0.6. According to Trio's semiring provenance we attach the following

lineage on each alternative:
$\lambda(31) = (11, 1) \wedge (21, 1)$
$\lambda(32) = (12, 1) \wedge (21, 1)$
$\lambda(33) = (11, 2) \wedge (21, 2)$.
Similarly in the answer of query Q_2 we expect:
$\mathbf{Q_2}(\mathbf{Q_1}(\mathbf{U}))$={41<Hank:0.8>:0}
$\lambda(41) = \{((11, 1) \wedge (21, 1)) \vee ((12, 1) \wedge (21, 1)) \vee ((11, 2) \wedge (21, 2))\}$
We would like to be able to directly compute those answers of Q_1 and Q_2 without having to compute all (exponentially many) possible worlds. As we already mentioned, existing work about possibilistic theory, join or projection with duplicate elimination was not possible due to the fact that possibistic sets were not powerful enough to express the disjunction of two different tuples occuring in the answer [BP05]. For example possibility theory could not model the fact that, e.g., tuples 31 and 33 in the answer of Q_1 could not coexist. Provenance poses additional logical restrictions to where an alternative can exist, thus overcoming this obstacle.

On the other hand until now provenance has only been combined with probabilistic theory and not with possibilistic. But probabilities have high complexity: for example if we want to compute the probability of alternative 41 *Hank* we must compute the probability of $\{((11, 1) \wedge (21, 1)) \vee (12, 1) \wedge (21, 1) \vee ((11, 2) \wedge (21, 2))\}$. In general computing the probability of a DNF boolean formula is #-P complete [BSH+08a, DS04, RPT11]. In contrast in our model which uses possibilities we can compute answers of selection with equality, projection and join in polynomial time. Note in particular that the possibility of the union of two events is always equal to the maximum of their possibilities. We use provenance only to restrict data. The computation of possibilities and necessities is not based on provenance; instead, they are computed directly from initial data. Provenance (which includes only possibilities of alternatives and not necessities of x-tuples) is inadequate of computing x-tuple necessities.

5.3 The Operators

In this section we give the definitions of selection, projection and join operators. These definitions enable us to directly compute the answers of SPJ queries posed over an uncertain database of our model with x-tuples and possibilities without having to compute first its possible worlds. In addition the computation of the possibilities of the answers is polynomial.

Let r be an uncertain relation of our model, A an attribute and $(A = q)$ a logical selection condition where q can be another attribute or a constant.

With $alt(t)$ we denote the alternatives of x-tuple t:

Selection
select$(r, A = q) = \{< restict(alt(t), A = q) >: N'$
such that $t{:}N \in r$ and where:
$N'{=}\min\{1 - \max_{t'_i \in alt(t) \wedge t'_i \not\models (A=q)} \{\Pi(t'_i)\}, N(t)\}$ and:
$restrict(alt(t), A = q) =$
$\{t'{:}\Pi,\ \lambda(t')$ such that: $t' \in alt(t)$, where $t \in r$, and $t' \models (A = q)$ and $\Pi = \Pi(t')$ and $\lambda(t') = Id(t')\}$.

We keep in the select result only the alternatives that satisfy our select condition, with the same possibility that they had in our initial database. We set as their lineage, the lineage pointing to the identifiers of the initial alternatives. As for necessity of each resulting x-tuple, it is the minimum of: i) the necessity of the original x-tuple they belonged to, ii) the initial necessity of the original alternatives and iii) of 1 minus the maximum possibility that an alternative of the same original x-tuple that does *not* satisfy our selection condition has. The proof that our system is closed under selection with equality conditions uses a combination of the closure of Trio system [BSH+08a] and the closure of selection on Possibilistic databases [BP05].

Projection
project$(r, X) = \{< t'.X : \Pi' >: N'$
such that: $t \in r$ and $t' \in alt(t)$ and $N' = \max_{A_i}\{N_i\}$ where: $A_i = \{N_i \mid t_i : N_i \in r$ and $\exists t'_i \in alt(t_i)$ with $t'_i.X = t'.X\}$
and $\Pi' = \max_{B_i}\{\Pi_i\}$ where:
$B_i = \{\Pi_i \mid t'_i{:}\,\Pi_i$ where $t'_i \in alt(t_i)$ and $t_i \in r$ and $t'_i.X = t.X\}$.
We also set: $\lambda(t'.X) = \vee_{C_i} id(t'_i)$ where $C_i = \{id(t'_i) \mid t'_i \in alt(t_i)$ where $t'_i \in alt(t_i)$ and $t_i \in r$ and $t'_i.X = t.X\}$.

We project the set of attributes X from every alternative and we perform alternative duplicate elimination. Thus the necessity of each resulting x-tuple is equal to the maximum necessity of each original x-tuple that includes an alternative that has the same projected value as the one alternative of our resulting x-tuple has. The same holds for the new possibility as well. Finally we set as lineage the disjunction of alternatives that give the same projected value. The proof that our system is closed under projection is easy. Let us just mention that the use of lineage allows duplicate elimination without losing the correct possible worlds. In addition for the possibilities we can easily use the maximum for the union of two alternatives with same data when we perform duplicate elimination.

Join

join$(r_1, r_2, A = B) = \{restrict(alt(t_1) \oplus alt(t_2), A = B)$
such that: $t_1 : N_1 \in r_1$ and $t_2 : N_2 \in r_2$ where:
$restrict(alt(t_1) \oplus alt(t_2), A = B) =$
$< t'_1 \oplus t'_2 : \Pi', \lambda'(t'_1 \oplus t'_2) >: N'$ such that: $t_1 \in r_1$ and $t'_1 \in alt(t_1)$ and $t_2 \in r_2$ and $t'_2 \in alt(t_2)$ and $t'_1 \oplus t'_2 \models (A = B)$ and $\Pi' = \min\{\Pi(t'_1), \Pi(t'_2)\}$ and
$N' = min\{1 - \max\limits_{t1''_i \in alt(t_1)/t'_1} \{\Pi(t''_{i1})\},$
$1 - \max\limits_{t''_{i2} \in alt(t_2)/t'_2} \{\Pi(t''_{i2})\}, N_1, N_2\}$ and
$\lambda'(t'_1 \oplus t'_2) = id(t'_1) \wedge id(t'_2)$.

We note that $\oplus$ denotes the concatenation of tuples. Also the above definition can be easily adopted to the case where the join condition involves a conjunction of attribute equalities. We restrict in the join results only the tuples that satisfy the join condition and we perform duplicate elimination on alternatives. The new possibility of each alternative of the result is equal to the minimum of the possibilities of the original alternatives that contributed to its value. The necessity of each resulting x-tuple is the minimum of: i) 1 minus the maximum possibility of each other alternative that exists in the original contributing x-tuples and ii) the necessities of the original contributing x-tuples. We also set as lineage the conjunction of the initial contributing alternatives.

We now show that our system with Trio's lineage, x-tuples, possibilities and necessities is closed for the join operation. Moreover it follows from the definitions of our operators that their complexity is polynomial to the size of the data (alternatives) of our initial uncertain database.

Theorem 1: The possibilistic database model with provenance is *closed* under the join operation.

Proof: We want to prove that $PW($join$(r_1, r_2, A = B)) =$ join$(PW(r_1, r_2), A = B)$. Suppose that r_1 and r_2 both contain a single x-tuple. So suppose that r_1 has x-tuple $t_1{:}N_1$ and r_2 has x-tuple $t_2{:}N_2$. Note that we have no loss of generality: The possibilities that alternatives have in every x-tuple in base relations form a possibilistic distribution. As a result in every base x-tuple always exists (at least one) alternative with possibility equal to 1. Suppose now that t' is an alternative in the result of a join query, resulting from two alternatives t'_1 of t_1 and t'_2 of t_2. The possibility of t' in the join result according to our definition is equal to the minimum of possibilities of t'_1 and t'_2. If r_1 and r_2 had more x-tuples then t'_1 and t'_2 would exist in more possible worlds resulting from the choices of alternatives from the other x-tuples. According to our semantics in the join result t' should have the possibility of the union of all its occurrences in every possible world. From the definition of possibility union this would be equal to the maximum of the possibilities of all possible worlds in which t'_1 and t'_2 both exist. But the maximum possibility exists in the possible world where t'_1 and t'_2 are selected from t_1 and t_2 and for all the other x-tuples the alternative with possibility equal to 1 has been selected. Hence the possibility of this possible world is equal to $\min\{\Pi(t'_1), \Pi(t'_2), 1, \ldots, 1\} = \min\{\Pi(t'_1), \Pi(t'_2)\}$, so equal to the case where r_1 and r_2 had only one x-tuple.

So suppose that r_1 has x-tuple $t_1{:}N_1$ and r_2 has x-tuple $t_2{:}N_2$. We remind that with $alt(t_1)$ we denote the set of alternatives that exist in x-tuple t_1 (respectively for t_2). We first show that $PW(\text{join}(r_1, r_2, A = B)) \subset \text{join}(PW(r_1, r_2), A = B)$. Let W_k be a possible world of $PW(\text{join}(r_1, r_2, A = B))$ and π_k its possibility. We want to show that W_k is also a world of $\text{join}(PW(r_1, r_2), A = B)$ with the same possibility. We consider two cases:

• $W_k \neq \emptyset$: We denote with t' an arbitrary alternative in W_k. From the definition of join the data of t' comes from the concatenation of two alternatives t'_1 of x-tuple t_1 and t'_2 of t_2 that satisfy the join condition (i.e., $t' = t_1 \oplus t_2$). These two alternatives also exist in a $PW(r_1, r_2)$, let us denote it with W'_k [2]. On the other hand if there exists a combination of two alternatives of x-tuples of r_1 and r_2 in a possible world W'_k of $PW(r_1, r_2)$ that satisfy the join condition then the join answer resulting from them appears in $\text{join}(PW(r_1, r_2), A = B)$. So there exists a possible world exactly equal to W_k as concerns data (and with lineage pointing to the same base data) in $PW(\text{join}(r_1, r_2, A = B))$. Note that as concerns data (and lineage) a similar result was also proven in [BSH+08a].

Now for the possibilities of W_k and W'_k: Again let t' be an arbitrary alternative in $W_k \in PW(\text{join}(r_1, r_2, A = B))$. As we just showed, a tuple with same data also appears in $W'_k \in \text{join}(PW(r_1, r_2), A = B)$. Its possibility in W'_k is associated with the possibilities of $t'_1 \in PW(r_1)$ and $t'_2 \in PW(r_2)$. Specifically it is equal to the minimum of possibilities of t'_1 and t'_2 since a possible world that produces t' must include them both (conjunction of possibilities). The same choices have been made in W_k to derive t' and according to our join definition the possibility degrees of the join query is equal to the minimum of possibilities of alternatives that produce the result. So the possibilities are the same in W'_k and W_k.

• $W_k = \emptyset$: We can have two subcases: either $\text{join}(r_1, r_2, A = B)$ is empty (t' does not exist) or the necessity N' of t' is less than 1. If it is empty then $(PW(r_1, r_2), A = B)$ is also empty and the possibility of the empty world is

[2]Unless they have extraneous lineage, but in that case they also not exist in $PW(\text{join}(r_1, r_2, A = B)$, we refer to [BSH+08a] for more details).

in both cases equal to the maximum possibility of any possible world, i.e., : $\min\{\max_{t''_{i1} \in alt(t_1)}\{\Pi(t''_{i1})\}, \max_{t''_{i2} \in alt(t_2)}\{\Pi(t''_{i2})\}\}$.

If join$(r_1, r_2, A = B)$ is not empty then the necessity degree N' of t' is less than 1 and the empty world W_k has possibility $1 - N'$. The possibility of W_k must correspond to a world of $PW(r_1, r_2)$ with the most possible choices of alternatives t'_1 of r_1 and t'_2 of r_2 that do not satisfy the join condition, i.e., with possibility: $\max\{\max_{t1''_i \in alt(t_1)/t'_1}\{\Pi(t''_{i1})\}, \max_{t''_{i2} \in alt(t_2)/t'_2}\{\Pi(t''_{i2})\}, 1 - N_1, 1 - N_2\}$. From our definition of join we see than indeed this would be the possibility of the empty world W_k. Using a similar logic it is now easy to also prove that join$(PW(r_1, r_2), A = B) \subset PW($join$(r_1, r_2, A = B))$.

5.3.1 Examples of Operators

We present in this subsection that if our join and project operators are posed over the initial data of our running example, their result directly computes the results $Q_1(U)$ and $Q_2(Q_1(U))$ with the correct data and provenance that we expected and presented in subsection 5.2.3.

Join

We illustrate the use of our operators join and project though our running example. Query Q_2 is a projection of attribute *person* on the result of query Q_1 which is the join of *Saw* and *Drives* over common attribute *car*. We begin with join query Q_1. According to our definition we have the query *join*(*Saw*, *Drives*, *car* = *car*). In our result we naturally restrict the combinations of all possible alternatives of the two relations *Saw* and *Drives* to the ones that satisfy the condition *car* = *car*. In our example there exist three such combinations. For the first one we have: Alternatives $11, 1$ and $21, 1$ from x-tuples 11, with necessity $N_{11} = 1$ and 21 with necessity $N_{21} = 1$, yield alternative $(Amy, Mazda, Hank)$. According to our join definition its possibility is $\Pi' = \min\{\Pi(11, 1), \Pi(21, 1)\} = \min\{1, 0.6\} = 0.6$. Respectively its necessity is $N' = \min\{1 - \max\{\Pi(11, 2)\}, 1 - \max\{\Pi(21, 2)\}, N_{11}, N_{21}\} = \min\{1 - \max\{0.8\}, 1 - \max\{1\}, 1, 1\} = \min\{1 - 0.8, 1 - 1, 1, 1\} = \min\{0.2, 0, 1, 1\} = 0$. The lineage of $(Amy, Mazda, Hank)$ is equal to the conjunction of the identifiers of the alternatives that produce it, i.e., with $(11, 1) \wedge (21, 1)$. For the other two combinations of alternatives that satisfy our join condition, the procedure is similar. In order to succinctly denote lineage we attach a new fresh identifier to each tuple-alternative of the answer. The final result is:

$\mathbf{Q_1(U)}$={31<Amy, Mazda, Hank:0.6>:0,
32<Billy, Mazda,Hank:0.4>:0
33<Amy, Toyota,Hank:0.8>:0}
$\lambda(31) = (11,1) \wedge (21,1)$
$\lambda(32) = (12,1) \wedge (21,1)$
$\lambda(33) = (11,2) \wedge (21,2)$

Projection

We continue with query Q_2 which is a projection of attribute $person$ from the result $Q_1(U)$, i.e., $project(Q_1(U), person)$. Our result has a singe x-tuple with one alternative $Hank$. Its necessity, according to our definition is equal to the maximum of the necessities of the x-tuples that produce the same result $Hank$ (we perform duplicate elimination). In our case all x-tules 31, 32 and 33 produce $Hank$, so the set A_i includes the necessities of all of them. So the necessity of $Hank$ in the result is: $N' = \max\{N(31), N(32), N(33)\} = \max\{0\} = 0$. The possibility of $Hank$ is equal to the maximum of the possibilities of all the alternatives that give result $Hank$. In our case the set B_i of such alternatives includes $(31,1)$, $(32,1)$ and $(33,1)$. So the possibility of $Hank$ in the result is: $\Pi' = \max\{\Pi(31,1), \Pi(32,1), \Pi(33,1)\} = \max\{0.4, 0.8, 0.6\} = 0.8$. The lineage of $Hank$ is equal to the disjunction of the identifiers of the alternatives that have $Hank$ in the data of the projection result. In order to succinctly denote lineage we attach new identifier 41 to x-tuple $Hank$ of the answer. So we have: $\lambda(41) = \{(31,1) \vee (32,1) \vee (33,1)\}$. As noted in [BSH+08a] we can use the lineage information of the initial $Q_1(U)$ and expand with polynomial complexity lineage back to base data. Hence we can replace, e.g., $(31,1)$ with its lineage in $Q_1(U)$, which is: $\{(11,1) \wedge (21,1)\}$. The final result is:

$\mathbf{Q_2(Q_1(U))}$={41<Hank:0.8>:0}
$\lambda(41) = \{((11,1) \wedge (21,1)) \vee ((12,1) \wedge (21,1)) \vee ((11,2) \wedge (21,2))\}$

Chapter 6

Efficient Lineage for SUM Aggregate Queries

AI systems typically make decisions and find patterns in data based on the computation of aggregate and specifically sum functions, expressed as queries, on data's attributes. This computation can become costly or even inefficient when these queries concern the whole or big parts of the data and especially when we are dealing with big data. New types of intelligent analytics require also the explanation of why something happened.

In this chapter we present a randomised algorithm that constructs a small summary of the data, called Aggregate Lineage, which can approximate well and explain all sums with large values in time that depends only on its size. The size of Aggregate Lineage is practically independent on the size of the original data. Our algorithm does not assume any knowledge on the set of sum queries to be approximated.

6.1 Introduction

Big data poses new challenges not only in storage but in intelligent data analytics as well. Many organisations have the infrastructure to maintain big structured data and need to find methods to efficiently discover patterns and relationships to derive intelligence [ora13a, ora13b]. Thus, it would be desirable to be able to construct out of big data a right representative part that can explain aggregate queries, e.g., why the salaries or the sales of a department are high.

AI systems typically make decisions based on the value of a function computed on data's attributes. Several approaches have in common the computation of aggregates over the whole or large subsets of the data that helps explain patterns and trends of the data. E.g., recommendation systems rank and retrieve items that are more interesting for a specific user by aggregating existing recommendations [RRSK11]. For another example, collaborative filtering computes a function which uses aggregates and a sum over the existing ratings from all users for each product in order to predict the preference of a new user [BHK98, KvdLvW09]. User preferences are often described as queries [Jan09], e.g., queries that give constraints on item features that need to be satisfied.

Another reason for which data analytics seek to explain data is for data debugging purposes. *Data debugging*, which is the the process that allows users to find incorrect data [MGNS11, MBM13], is a research direction that is growing fast. Data are collected by various techniques which, moreover, are unknown to and uncontrolled by the user, thus are often erroneous. Finding which part of the data contains errors is essential for companies and affects a large part of their business.

All these applications call for techniques to explain our data. Aggregation is a significant component in all of them. In this chapter we offer a technique that constructs a summary of the data with properties that allow it to be used efficiently to explain much of the data behaviour in aggregate for sums. We refer to this summary as *Aggregate Lineage*, since in most applications it represents the source of an aggregate query[1].

Lineage (a.k.a. provenance) keeps track of where data comes from. Lineage has been investigated for data debugging purposes [ICF+12]. Storing the complete lineage of data can be prohibitively expensive and storage-saving techniques to eliminate or simplify similar patterns in it are studied in [CJR08]. For select-project-join SQL queries, lineage stores the set of *all* tuples that were used to compute a tuple in the answer of the query [BSH+08a]. This is natural for select-project-join SQL queries where original attribute values are "copied" in attribute values of the answer. However, in an aggregate query the value of the answer is the result of applying an aggregate function over many numerical attribute values. When we want to understand why we get an aggregate answer it may no longer be important or feasible to have lineage to point to all contributing original tuples and their values. We would rather want to compute *few* values that can be used to tell us as much as possible about the origin of the result of an aggregate query. However is this at all possible and if it is what are the limitations?

In this chapter we initiate an investigation of such questions and, interestingly, we show that useful and practical solutions exist. In particular, we offer a technique that uses randomisation to compute Aggregate Lineage

[1]Lineage used to be referred to as "explain" in database papers of the late 80's.

which is a small representative sample (it is more sophisticated than just a simple random sample) of the data. This sample has the property to allow for good approximations of a sum query on ad hoc subsets of data – we call them *test queries*. Test queries are applied to the Aggregate Lineage – not the whole original data. The test queries which we consider are sum queries with same aggregated attribute conditioned with any grouping attributes depending on which subsets of the data we want to test. We give performance guarantees about the quality of the results of the test queries that show the approximation to be good for test queries with large values (i.e., close to the total sum over the whole set of data). Our performance guarantees hold, with high probability, for any set of queries, even if the number of queries is exponentially large in the size of the lineage. The only restriction is that the queries should be oblivious to the actual Aggregate Lineage. This restriction is standard in all previous work on random representative subsets for the evaluation of aggregate queries and is naturally satisfied in virtually all practical applications. The following example offers a scenario about how Aggregate Lineage can be used in data debugging and demonstrates how some test queries can be defined.

Example 26. Suppose that the accounting department of a big company maintains a database with a relation *Salaries* with hundreds of attributes and millions of tuples. Each tuple in the relation may contain an identifier of an employee stored in attribute *EmplID*, his Department stored in attribute *Department*, his annual salary stored in attribute *Sal* and many more attribute values. Other relations are extracted from this relation, e.g., a relation which contains aggregated data such as the total sum of salaries of all employees. A user is trying to use the second relation for decision making but he finds that the total sum of salaries is unacceptably high. He does not have easy access to the original relation or he does not want to waste time to pose time-consuming queries on the original big relation. The error could be caused by several reasons (duplication of data in a certain time period, incorrect code that computes salaries in a new department). Thus e.g., if we could find the total sum of salaries for employees in the toy department during 2009, and see that this is unreasonably high, still close to the first total sum of all employees' salaries, then we will be able to detect such errors and narrow them down to small (and controllable) pieces of data.

In order to do that, we need the capability of posing sum queries restricted to certain parts of the data by using combinations of attributes. This will help the user understand which piece of data is incorrect. We do not know in advance, however, which piece of data the user would want to inquire and thus Aggregate Lineage should allow the user to be able to get good approximated answers to whatever queries he wants to try. There are billions of such possible queries and hence billions of subsets of data which we want to compute a good approximation of the summation of salaries. We want Aggregate Lineage to offer this possibility.

We propose to keep as Aggregate Lineage a small relation under the same schema of the original relation. In order to select which tuples to include, we use valued-based sampling with repetition, i.e., weighted random sampling where the probability of selecting each tuple is proportional to its value on the summed attribute. The intuition why this method works is the following. Larger values contribute more to the sum than smaller ones, thus we expect that tuples with larger values should be selected more often than tuples with smaller values. Hence, we could end up with a tuple selected many times in the sample even if it appears only once in the original data. On the other hand, if there are many tuples with values of moderate size, many of them will be selected in the Aggregate Lineage, so that their total contribution to the approximation of the sum remains significant.

6.1.1 Our contribution

In our approach Aggregate Lineage is a small relation with same schema as the original relation and with the property to offer good approximations to test queries posed on it.

To present performance guarantees, we build on Althöfer's Sparsification Lemma [Alt94]. In [Alt94], Althöfer shows that the result of weighted random sampling over a probability vector is a sparse approximation of the original vector with high probability. This technique has found numerous applications e.g., in the efficient approximation of Nash equilibria for (bi)matrix games [LMM03], in the very fast computation of approximate solutions in Linear Programming [LY94], and in selfish network design [FKS12].

In this chapter, we show for the first time that the techniques of [Alt94] are also useful in the context of sum database queries with lineage. Our results show that the Aggregate Lineage that we extract has the following properties (which we describe in technical terms and prove rigorously in Section 6.4):

- Its size is practically independent of the size of the original data.
- It can be used to approximate well all "large" sums (i.e., with values close to the total sum), of the aggregated attribute in time that depends only on its size, and thus is almost independent of the size of the original data.

$t[A]$	value of attribute A in tuple t
S	sum of all values over attribute A
p_t	probability that Algorithm Comp-Lineage selects tuple t
Fr	additional attribute recording the frequency of a tuple in the lineage
$L_{R.A}$	the lineage relation computed by Algorithm Comp-Lineage wrto attribute A
$Q(R.A)$ (Sec. 4)	a sub-sum query computed over original relation R wrto A
$Q'(L_{R.A})$ (Sec. 4)	sub-sum query Q computed over the aggregate lineage relation
I_R^Q (Sec. 4)	set of identifiers of the tuples in relation R that satisfy the predicates in query Q

Figure 6.1: Main symbols used in this chapter.

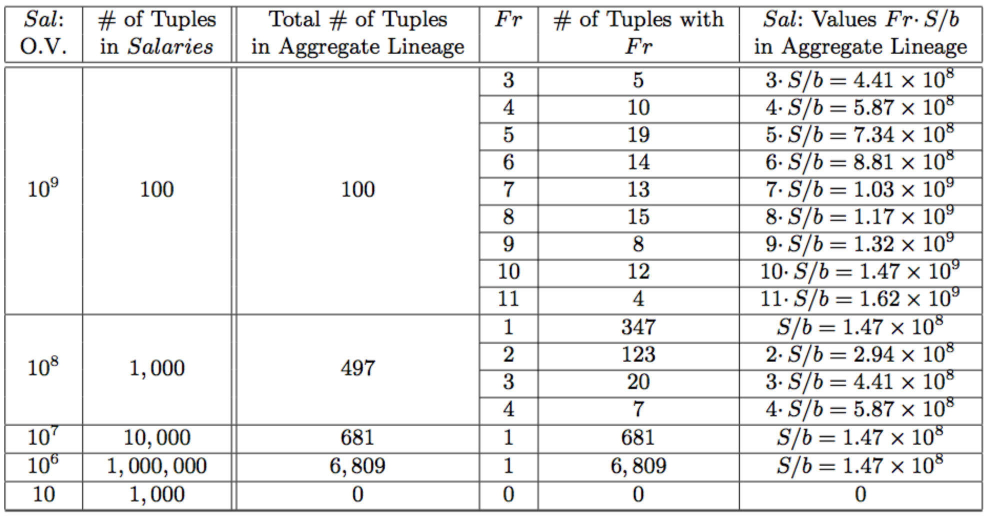
6.2 Computing Aggregate Lineage

In this section, we present randomised algorithm Comp-Lineage which computes Aggregate Lineage in one pass over the data and in time linear in the size n of the original database relation. In Section 6.4 we show that the output of Comp-Lineage is useful to approximate *arbitrary* ad-hoc sum test queries in time independent of n. We note that our algorithm is agnostic of the specific sum queries that will be approximated by using its output.

Suppose that we are given a database with a relation R with n tuples and we are given a positive integer b which is the number of tuples we have decided to include in the Aggregate Lineage (in Section 6.4 we will explain how we decide b to give good performance and approximation guarantees). Suppose that A is a numerical attribute of R which takes nonnegative values. Let S be the sum of values of attribute A over all n tuples. The algorithm essentially is a biased sampling with repetition that selects b tuples from R. Each tuple t has probability to be selected equal to $p_t = t[A]/S$ where $t[A]$ is the value of attribute A in t. It collects initially a bag (a.k.a. multiset and is allowed to have the same element more than once) of tuples (since each tuple may be selected multiple times) which is turned in a set of tuples by adding an extra attribute Fr (for Frequency) which shows the number of times this tuple is selected. We denote by $L_{R.A}$ the Aggregate Lineage of relation R with sum attribute A.

Algorithm Comp-Lineage
Input: A relation R with n tuples and positive integer $b < n$.
Output: An Aggregate Lineage relation $L_{R.A}$ with at most b tuples.

- Randomly select with repetition one out of the n tuples of R in b trials where each tuple t is selected with probability p_t.
- Form relation $L_{R.A}$ by including all tuples selected above and adding an extra attribute Fr to each tuple to record how many times this tuple was selected.



Sal: O.V.	# of Tuples in *Salaries*	Total # of Tuples in Aggregate Lineage	*Fr*	# of Tuples with *Fr*	*Sal*: Values $Fr \cdot S/b$ in Aggregate Lineage
10^9	100	100	3	5	$3 \cdot S/b = 4.41 \times 10^8$
			4	10	$4 \cdot S/b = 5.87 \times 10^8$
			5	19	$5 \cdot S/b = 7.34 \times 10^8$
			6	14	$6 \cdot S/b = 8.81 \times 10^8$
			7	13	$7 \cdot S/b = 1.03 \times 10^9$
			8	15	$8 \cdot S/b = 1.17 \times 10^9$
			9	8	$9 \cdot S/b = 1.32 \times 10^9$
			10	12	$10 \cdot S/b = 1.47 \times 10^9$
			11	4	$11 \cdot S/b = 1.62 \times 10^9$
10^8	1,000	497	1	347	$S/b = 1.47 \times 10^8$
			2	123	$2 \cdot S/b = 2.94 \times 10^8$
			3	20	$3 \cdot S/b = 4.41 \times 10^8$
			4	7	$4 \cdot S/b = 5.87 \times 10^8$
10^7	10,000	681	1	681	$S/b = 1.47 \times 10^8$
10^6	1,000,000	6,809	1	6,809	$S/b = 1.47 \times 10^8$
10	1,000	0	0	0	0

Figure 6.2: Properties of Aggregate Lineage $L_{Salaries.Sal}$ for $b = 8,852$. The first two columns describe the data. The next three columns describe the Aggregate Lineage relation. The last column shows how we use this lineage to compute sub-sums.

We can use the techniques of [ES06] for weighted random sampling and efficiently implement our algorithm to run in linear time in the size of the input either in a parallel/distributed environment or over data streams.

Table 6.1 summarizes the main symbols used throughout the chapter.

6.3 Running Example

Example 27. We illustrate Algorithm Comp-Lineage by applying it to Example 26 with $b = 8,852$ and presenting the data and the Aggregate Lineage in Figure 6.2. Actually Figure 6.2 only shows the value of the aggregated attribute (*Sal* in our example), the rest of the tuple is not shown.

The first two columns of Figure 6.2 present the data in relation *Salaries*. In order to be able to present many tuples we have chosen a relation with a few values for attribute *Sal*, actually five (i.e., $10^9, 10^8, 10^7, 10^6$ and 10) and their Original Values (O.V.) are shown in the first column. The second column shows how many tuples in *Salaries* have these values in *Sal*. Thus, it says, e.g., that there are 100 tuples with value in *Sal* equal to 10^9, 1,000 tuples with value in *Sal* equal to 10^8 and so on.

The third column in Figure 6.2 shows how many tuples from *Salaries* with a specific value in *Sal* are selected by Algorithm Comp-Lineage to be included in the Aggregate Lineage relation. Thus, e.g., all 100 tuples with $Sal = 10^9$ were chosen, only 681 tuples with $Sal = 10^7$ were chosen and no tuple with $Sal = 10$ was chosen.

In order to represent the Aggregate Lineage relation $L_{Salaries.Sal}$ in the most demonstrative way, we have chosen to partition its tuples in blocks (each block further divided in multiple rows in columns 4, 5 and 6), each block corresponding to one value of Sal in $Salaries$. Thus the first block has 9 rows, the second block has 4 rows and the last three blocks have one row each. This breaking into blocks gives a visualisation of the characteristics of the algorithm.

The fourth column stores the extra attribute frequency Fr which tells how many times a certain tuple was selected by the algorithm and the fifth column stores the number of tuples that were selected so many times. Thus, e.g., the first row says that 5 tuples were selected 3 times each. The ninth row says that 4 tuples from $Salaries$ were selected 11 times each.

The blocks give us an intuition of the characteristics of the Aggregate Lineage. The first block corresponds to the largest value of Sal and tuples with this value (i.e., $Sal = 10^9$) contributed quite heavily to the lineage - all 100 tuples with $Sal = 10^9$ were selected multiple times. In more detail, there are 100 tuples with value $Sal = 10^9$. Of those tuples, 5 were added in the bag 3 times each, 10 tuples were added in the bag 4 times each, and so on. Thus, by considering these 100 tuples, the Algorithm Comp-Lineage added in the bag $3 \cdot 5 + 4 \cdot 10 + 5 \cdot 19 + 6 \cdot 14 + 7 \cdot 13 + 8 \cdot 15 + 9 \cdot 8 + 10 \cdot 12 + 11 \cdot 4 = 681$ tuples in total. That is to say, each of those 100 tuples contributed on average 6.81 to the bag. When we get a set out of the bag by using frequencies (to avoid repeating a tuple multiple times), then we see that the average frequency per tuple is 6.81. So, from this first block, the 681 tuples in the bag of Algorithm Comp-Lineage are transformed to a set of 100 tuples in Aggregate Lineage with average frequency 6.81. We can compare it with the average frequency in the second block which is 0.681 (this is $1 \cdot 347 + 2 \cdot 123 + 3 \cdot 20 + 4 \cdot 7 = 681$ divided by 1000 tuples) and see that, in the data of our example, each tuple of the first block contributes more heavily to the lineage.

As we will explain in more detail later, this shows partly why the lineage is useful for discovering almost accurately sub-sums that are large compared to the total sum, whereas when a sub-sum is small in comparison, then the lineage cannot be used to compute it accurately.

The second block did not contribute that heavily but still quite a lot, around half of tuples with $Sal = 10^8$ were selected at least once and quite a few more than once, in total this block contributed 681 tuples in the bag. The third block contributed moderately. The fourth block is interesting because the value of Sal is very small only 10^6 but it contributed quite a lot due to the fact that there are many tuples in $Salaries$ with $Sal = 10^6$, thus it contributed almost 85 percent of the tuples in the Aggregate Lineage.

Finally the last column in the figure shows how much each tuple from the

Aggregate Lineage contributes to the approximation of sub-sums that are computed by the test queries. The same tuple is added in the Aggregate Lineage several times as recorded in the new attribute Fr and thus, in order to calculate the contribution of a certain tuple, we multiply its frequency in Fr by S/b. By doing so, some tuples (e.g., the ones in the fourth block) in our example of Figure 6.2 will contribute much more than their actual value in Sal. But this is to compensate for the tuples with value close to it (same value in our example) that are not selected to be included in the Aggregate Lineage. In the next section we give the technical details on how Aggregate Lineage can be used in order to approximate sub-sums.

Note that Aggregate Lineage does not assume any knowledge of the query set: i.e., we run the random selection of Algorithm Comp-Lineage only once and compute $\mathrm{L}_{R.A}$ without assuming anything about the queries. Then, this same relation $\mathrm{L}_{R.A}$ can be used to make us understand any sub-sum test query, without requiring that the test queries are given beforehand or requiring that the test queries are chosen in any specific fashion (e.g., they do not have to be chosen uniformly at random), as long as the query choice is oblivious to the actual sample computed by Aggregate Lineage[2]. We first present the theoretical approximation guarantees and then demonstrate how these guarantees play for debugging on our running example.

6.4 Approximation Guarantees of Test Queries on Aggregate Lineage

In this section we prove the theoretical guarantees of Aggregate Lineage. Let R be a relation with a nonnegative numerical attribute A. We consider SUM queries that ask for the sum of attribute's A values over arbitrary subsets of the tuples in relation R. We use tuple identifiers in order to succinctly represent subsets of tuples. Thus, any SUM query defines a set of tuple identifiers for tuples that satisfy its predicates, hence the following formal definitions:

Definition 46 (Exact SUM $Q(R.A)$)**.** Let R be a database relation. We attach a tuple identifier on each tuple of R. We denote by I_R the set of all identifiers in relation R. Given an attribute A in the schema of R, we denote by a_i the value of attribute $R.A$ in the tuple with identifier i in R.

Let Q be a SUM query over $R.A$. We denote by I_R^Q the set of tuple identifiers from I_R for tuples of R that satisfy Q's predicates.

[2]In technical terms, the queries are posed by an *oblivious adversary*, i.e., an adversary that knows how exactly Aggregate Lineage works but does not have access to its random choices. The restriction to oblivious adversaries is standard and unavoidable, since if one knows the actual value of $\mathrm{L}_{R.A}$, he can construct a query that includes only tuples not belonging to $\mathrm{L}_{R.A}$, for which no meaningful approximation guarantee would be possible.

The result of a SUM query, $Q(R.A)$, is the summation of the values of $R.A$ over the set of tuples with identifiers that appear in I_R^Q, i.e., $Q(R.A) = \sum_{i \in I_R^Q} a_i$.

Definition 47 (Approximated SUM $Q'(L_{R.A})$)**.** Let Q be a SUM query over $R.A$ and let $L_{R.A}$ be an *Aggregate Lineage*. We attach a tuple identifier on each tuple of $L_{R.A}$. We denote by I_L the set of all identifiers in $L_{R.A}$. We denote by I_L^Q the set of tuple identifiers from I_L for tuples of $L_{R.A}$ that satisfy Q's predicates (since the set of attributes of R is a subset of the set of attributes of $L_{R.A}$, we have that the predicates of a SUM query Q, expressed on attributes of R, define I_L^Q).

We denote by f_i the value of attribute $L_{R.A}.Fr$ in the tuple with identifier i in $L_{R.A}$.

The approximated result of SUM query Q, denoted by $Q'(L_{R.A})$, is the summation of the values of $L_{R.A}.Fr$ over the set of tuples with identifiers that appear in I_L^Q multiplied by S/b, i.e., $Q'(L_{R.A}) = \sum_{i \in I_L^Q} f_i \cdot S/b$.

The following theorem provides the performance guarantees for any arbitrary set of m SUM queries computed over the Aggregate Lineage relation in order to serve as an approximation of the corresponding SUM queries over the original data.

Theorem 24. *Let R be a relation with n tuples having nonnegative values $a_1, \ldots, a_n$ on attribute A, and let $S = \sum_{i=1}^{n} a_i$. Then, for any collection of m SUM queries $Q_1(R.A), \ldots, Q_m(R.A)$ (not known to the algorithm), any $p \in (0,1)$, and any $\epsilon > 0$, the Algorithm Comp-Lineage with input all tuples of R and $b = \lceil \ln(2m/p)/(2\epsilon^2) \rceil$ derives an* Aggregate Lineage *$L_{R.A}$ such that $|Q_j(R.A) - Q'_j(L_{R.A})| \leq \epsilon S$, for all $j \in [m]$, with probability at least $1-p$.*

Proof. The proof is an adaptation of the proof of Althöfer's Sparsification Lemma [Alt94]. For simplicity, we assume, without loss of generality, that the set I_R of all tuple identifiers of R in Definition 46 is $I_R = \{1, \ldots, n\}$. We define b independent identically distributed random variables $X_1, \ldots, X_b$, which take each value $i \in [n]$ with probability a_i/S. Namely, each random variable X_i corresponds to the outcome of the i-th trial of Comp-Lineage. For each tuple i, its frequency in the sample is $f_i = |\{k \in [b] : X_k = i\}|$.

Let us fix an arbitrary SUM query $Q_j(R.A)$. For each $k \in [b]$, we let Y_j^k be a random variable that is equal to 1, if $X_k \in I_R^{Q_j}$, and 0, otherwise. Since the random variable Y_j^k is equal to 1 with probability $Q_j(R.A)/S$, $\mathbb{E}[Y_j^k] = Q_j(R.A)/S$. We observe that the random variables $\{Y_j^k\}_{k \in [b]}$ are independent, because the random variables $\{X_k\}_{k \in [b]}$ are independent. Furthermore, we let Y_j be a random variable defined as

$$Y_j = \frac{1}{b} \sum_{k=1}^{b} Y_j^k$$

By definition, $Y_j = \sum_{i \in I_R^{Q_j}} f_i/b = \sum_{i \in I_L^{Q_j}} f_i/b$, and thus we have that $Y_j = Q'_j(L_{R.A})/S$, i.e., Y_j is equal to the approximated result of the SUM query divided by S. Also, by linearity of expectation, $\mathbb{E}[Y_j] = Q_j(R.A)/S$.

Applying the Chernoff-Hoeffding bound, we obtain that for the particular choice of b, with probability at least $1 - p/m$, the actual value of Y_j differs from its expectation $Q_j(R.A)/S$ by at most ϵ, which implies that $Q'_j(L_{R.A})$ differs from $Q_j(R.A)$ by at most ϵS. Formally, by the Chernoff-Hoeffding bound [3],

$$\begin{aligned} &\mathbb{P}\mathrm{r}[|Q'_j(L_{R.A}) - Q_j(R.A)| > \epsilon S] \\ &= \mathbb{P}\mathrm{r}[|Y_j - Q_j(R.A)/S| > \epsilon] \\ &\leq 2\exp^{-2\epsilon^2 b} \leq p/m\,, \end{aligned}$$

where the last inequality follows from the choice of b.

Applying the union bound, we obtain that

$$\mathbb{P}\mathrm{r}[\exists j \in [m] : |Q'_j(L_{R.A}) - Q_j(R.A)| > \epsilon S] \leq p$$

which concludes the proof of the lemma. □

Example 28. Suppose, in our running example, we want to be able to answer with good approximation $m = 10^6$ queries. What are the guarantees that the theorem provides? The original data have $n \approx 10^6$ tuples. Suppose we select the number of tuples in the Aggregate Lineage to be $b \approx 9000$. Then the theorem says that, by setting $\epsilon = 0.04$, we can compute any of 10^6 arbitrary queries within $0.04S$ of its real value with probability $1 - 10^{-6}$. Thus, if the real exact value of the query Q_1 is equal to $Q_1(Salaries.Sal) = 0.4S = S_1$ (remember S is the sum over all tuples of relation R) then the approximation will be $0.04S = 0.04S_1/0.4 = 0.1S_1$. If for another query Q_2 we have $Q_2(Salaries.Sal) = 0.8S = S_2$ then the approximation will be $0.05S_2$, so, then with high probability we get an answer that is within a factor of 0.05 of the actual answer.

Observations on the practical consequences of Theorem 24. Examining closely equation $b = \lceil \ln(2m/p)/2\epsilon^2 \rceil$ which gives us an upper bound of the number of tuples in the Aggregate Lineage for m queries and with p and ϵ guarantees as in its statement, we make the following observations:

- The value of b depends on m as the logarithm, hence if we go from m to m^2 queries, we only need to multiply b by 2 in order to keep the same performance guarantees. Thus it is reasonable to state that, in many practical cases the number m of queries that can be approximated well can be as large as a polynomial on the size of data – even with coefficient in the order of a few hundreds.

[3] We use the following form of the Chernoff-Hoeffding bound (see [Hoe63]): Let $Y^1, \ldots, Y^b$ be random variables independently distributed in $[0,1]$, and let $Y = \frac{1}{b}\sum_{k=1}^{b} Y^k$. Then, for all $\epsilon > 0$, $\mathbb{P}\mathrm{r}[|Y - \mathbb{E}[Y]| > \epsilon] \leq 2\exp^{-2\epsilon^2 b}$, where $\exp = 2.71\ldots$ is the basis of natural logarithms.

- The value of b does not depend much on p (again only as in the logarithm) but it depends mainly on ϵ which controls the approximation ratio (the approximation ratio itself is ϵ/ρ if the query to be computed has a sum $S' = \rho S$).

6.5 A debugging scenario

Here is what a user can do for data debugging when using the Aggregate Lineage we propose.

- He computes sub-sums by filtering some attributes and possibly specific values for these attributes. E.g., what was the sum of salaries of employees in the toy department in Spring 2010 and only for those employees who were hired after 2005. The user devises several such test queries as he sees appropriate and while he computes them and checks that sub-data is ok or suspicious, he devised different test queries to suit the situation. E.g., if he observes an unusually large value, close to the total sum, in the query about employees in the toy department and hired before 2005, then the rest of the queries he devises stay within this department and within the range until 2005, and tries to narrow down further the wrong part of data. E.g., now he narrows down to each month or/and to employees that are hired between 2005 and 2007, etc. On the other hand, if he finds the answer satisfactory, then he announces this part of the data correct, therefore stays outside this sub-data and tries to find some other part of the data that are faulty. The user uses and poses his test queries over the stored small Aggregate Lineage instead of inefficiently use the original big relation.

In the following example we show how using Aggregate Lineage to approximate test queries applies to our running example.

Example 29. We continue our running Example 27 where we computed Aggregate Lineage $L_{Salaries.Sal}$. Suppose that we have a SUM test query Q_1 asking the sum of the salaries of a subset of the employees of the company defined from a subset of *EmpID*'s. Let this subset consist of 50 employees with salary 10^9, $5,000$ employees with salary 10^7 (so half of them) and of all 10^6 employees with salary 10^6. We compute the query over *Salaries* and take the exact answer 1.1×10^{12}.

In order to use Aggregate Lineage to understand our data we compute $I_L^{Q_1}$. The Aggregate Lineage has at most $8,852$ tuples. The identifiers of $I_L^{Q_1}$ define the *sub-lineage* of query Q_1 over $L_{Salaries.Sal}$. The sub-lineage of Q_1 points to 50 of the tuples of $L_{Salaries.Sal}$ with original salaries 10^9 and to all $6,809$ tuples with original *Sal* values 10^6 (cf. Figure 6.2). It will also point to some tuples of $L_{Salaries.Sal}$ with *Sal* values 10^7: On average query Q_1 is applied on half of the 681 selected in Aggregate Lineage tuples, but in extreme cases it may include all or none of them. For this reason, it is a good practice to run the randomised algorithm more than once and compute a few distinct summaries in order to have better results. For instance, we may compute three summaries, use some benchmark sub-queries to decide a distance between summaries, toss the summary which is the more distant and keep one of the others arbitrarily. Note that it is easy to compute the benchmark queries in one pass through the original data in parallel with computing the lineage.

We now use the Aggregate Lineage $L_{Salaries.Sal}$ shown in Figure 6.2 to approximate the value of the sum answer to Q_1. In one worst case query Q_1 will include: the 50 tuples with salaries 10^9 from $L_{Salaries.Sal}$ tuples with the larger frequencies and all 681 selected tuples with salaries 10^7. The approximation $Q_1'(L_{Salaries.Sal})$ in this case is $(4 \cdot 11 + 12 \cdot 10 + \ldots + 681 + 6,809)S/b = 7,935 \cdot S/b = 1.17 \cdot 10^{12}$. In the other extreme case Q_1 includes tuples with the smaller frequencies and none of the selected in Aggregate Lineage tuples with salaries 10^7, yielding the approximation $6,995 \cdot S/b = 1.03 \cdot 10^{12}$. We see that Q_1 is well approximated. Of course the approximation bounds are not the same for every SUM query - we presented the guarantees in Section 6.4.

Another straw man approach would be to select as lineage the $8,852$ tuples with larger salary values. This method will select all 100 tuples with salaries 10^9, all $1,000$ tuples with salaries 10^8 and the remaining $7,752$ tuples from tules with salaries 10^7. With this approach, query Q_1 will be on average approximated with the value $50 \cdot 10^9 + 3,876 \cdot 10^7 \approx 8.8 \times 10^{10}$ because it loses all the information about all original 10^6 tuples with salaries 10^6 contributing to the sum. On another approach, a simple random sampling of $8,852$ tuples will almost always select all of them from the 10^6 many tuples with salaries 10^6. Query Q_1 will then be approximated with the value $8,852 \cdot 10^6 \approx 8.8 \times 10^9$. Note, on the other hand, that if all original tuples had the same salaries then our method would coincide with simple random sampling.

6.6 Discussion

We have focused in our exposition only on a single aggregated attribute (e.g., *Sal* in our example). This is done for simplicity. Our ideas can be easily extended to include more aggregated attributes as long as we are willing to keep a distinct aggregate lineage for each attribute. E..g., suppose we also had a *Rev* (for Revenue) attribute for each employee. In such a case we keep two lineage relations, one for *Sal* and one for *Rev*. The algorithm to compute them can be thought of as a parallel implementation of two copies of the algorithm Comp-Lineage. We need only one pass through the original data. The only difference is that now, a) we need the two total sums S_{Sal} and S_{Rev} and b) for each tuple t, we have two probabilities p_t^{Sal} and p_t^{Rev}, the first to be used for the lineage related to attribute *Sal* and the second to be used for the lineage related to attribute *Rev*.

Algorithm Comp-Lineage performs a weighted random sampling which selects with replacement b out of n tuples of R where the weight w_i for the tuple with identifier i is equal to the value of attribute A of this tuple. Using b copies (each copy selects a single element) of the weighted random sampling with reservoir algorithm presented in [ES06], we can implement Comp-Lineage in one-pass over R, in $O(bn)$ time and $O(b)$ space. This implementation can also be applied to data streams and to settings where the values of n and S are not known in advance.

However the technique in [ES06], does not seem to be efficiently parallelizable, at least not in a direct way. Thus the problem of how to efficiently implement our technique in distributed computational environments such as MapReduce remains open. Issues about how to implement sampling in MapReduce are discussed in [GC12]. Another open problem is how to apply this technique to evolving data [GR02]. In data streams, we assume that the sample is to be computed over the entire data. When data continuously evolve with time, the sample may also change considerably with time. The nature of the sample may vary with both the moment at which it is computed and with the time horizon over which the user is interested in. We have not investigated here how to provide this flexibility.

6.7 Comparison with Synopses for Data

There has been extensive research on approximation techniques for aggregate queries on large databases and data streams. Previous work considers a variety of techniques including random sampling, histograms, multivalued histograms, wavelets and sketches (see e.g., [CGHJ12] and the references therein for details and applications of those methods). Most of the previous work on histograms, wavelets, and sketches focuses on approximating aggregate queries on a given attribute A for specific subsets of the data that are known when the synopsis is computed (e.g., the synopsis concerns the entire data stream or a particular subset of the database). Thus, such techniques typically lose the correlation between the approximated A values and the original values of other attributes. For the more general case of multiple queries that can be posed over arbitrary sets of attributes and subsets of the data not specified when the synopsis is computed, those techniques typically lead to an exponential (in the number of other attributes involved) increase in the size of the synopsis (see e.g., [DGGR02, DGGR04]).

In contrast, our approach is far more general and does not focus on approximating queries over specific attributes or subsets of the data. Our algorithm computes a small sample without assuming any knowledge on the set of queries and keeps the association between the sampled A values and all other attributes. Then, we can use the Aggregate Lineage to approximate large-valued sum queries over arbitrary subsets of the data that can be expressed over any set of attributes. The Aggregate Lineage can approximately answer a number of queries exponential in its size. Of course, the queries should be oblivious to the actual Aggregate Lineage (technically, they should be computed by an oblivious adversary), but this technical condition applies to all previously known randomised synopses constructions (see e.g., [CGHJ12]).

Chapter 7

Optimizing 2-Way Skew Joins in MapReduce

Techniques have been proposed and implemented for skew joins in order to handle loss of parallelism when a small number of values of the join attribute(s) appear in a significant fraction of the tuples. However, these techniques do not minimize the *communication cost* – the amount of data transmitted from the mappers to the reducers. Communication cost is known to be an important parameter in MapReduce algorithms. In this chapter we propose a novel technique that minimizes the communication cost and show that it improves significantly on the techniques that have previously been implemented for handling skew in joins.

7.1 Introduction

Skew is a big concern when we design algorithms for parallel systems and in particular in MapReduce algorithms. Skew occurs when a small number of Map tasks and/or Reduce tasks do a significant fraction of the work. Since the work of a task cannot be divided among compute nodes, the entire job must take at least as much time as the longest task. For example, if one task requires 20% of the total time of all the tasks, then no matter how many compute nodes are used, the speed-up due to parallelism cannot be greater than a factor of 5.

Systems such as Pig or Hive that implement SQL or relational algebra over MapReduce have mechanisms to deal with joins where there is significant skew; i.e., values of the join attributes that appear very frequently (see, e.g., [ORS+08, TSJ+10, TSJ+09]). Both use a two-round algorithm, where the first round identifies the *heavy hitters*, those values of the join attribute(s) that occur in at least some given fraction of the tuples. In the second round, tuples that do not have a heavy-hitter for the join attribute(s) are handled normally. That is, there is one reducer[1] for each key, which is a value of the join attribute(s). This reducer handles all the tuples from both relations having that value for the join attribute(s). Since the key is not a heavy hitter, this reducer handles only a small fraction of the tuples, and thus will not cause a problem of skew.

Suppose some value b for attribute B is identified as a heavy hitter in the join of $R(A, B)$ with $S(B, C)$. Suppose there are r tuples of R with $B = b$ and there are s tuples of S with $B = b$. Suppose also for convenience that $r > s$; that is, the larger of the two sets of tuples with $B = b$ comes from R, and the smaller set comes from S. For any integer k we can hash values of attribute A to k buckets, using a hash function h, so the tuples of R that have $B = b$ will each be assigned to exactly one of these buckets. The buckets are the keys in a MapReduce job that handles only the value b. Each tuple $R(a, b)$ is sent to the bucket $h(a)$ only; that is, we *partition* the larger set of tuples with $B = b$. However, a tuple $S(b, c)$ is sent to all k of the buckets, so it can be joined with all the tuples of R that have $B = b$. We say the smaller set of tuples with $B = b$ is *replicated*.

In practice, there is not one MapReduce job for each heavy hitter. Rather, after the first job determines which values are heavy hitters, and how frequently they appear in each relation, it is a small matter to select an appropriate value of k for each heavy hitter (the k's need not be the same) and to decide, for each heavy hitter, which of the relations will have their tuples with this heavy-hitter value partitioned, while the other is replicated. Now a second MapReduce job can treat all the tuples of R and S appropriately. Tuples without a heavy-hitter value for B are handled normally; the B-value becomes the key. Tuples with a heavy-hitter value for B are either partitioned or replicated, depending on whether the tuple is part of the larger or smaller set of tuples with that value.

The approach described above appears not only in Pig and Hive, but dates back to [WDY93]. This work, which looked at a conventional parallel implementation of join rather than a MapReduce implementation, uses the same (highly non-optimal) strategy of choosing one side to partition and the other side to replicate.

However, these techniques do not optimize the communication cost, which is an important component of the total cost of a MapReduce algorithm – often the dominant cost. The *communication cost* is the total amount of data transmitted from the mappers to the reducers. The communication cost per input is the *replication rate*. These measures have been defined and studied in earlier work, including [ASSU13, ABS+12].

In [AU11], the *shares* technique is presented, which minimizes the communication cost for multiway joins. In the *shares* algorithm, keys are vectors of buckets. Here, we use the same idea to construct an algorithm for 2-way join that handles skew and gives a better communication cost than the algorithm described above; in fact, we show that the communication is minimized by our algorithm. Our

[1] In this chapter, we use the term *reducer* to mean the application of the Reduce function to a key and its associated list of values. It should not be confused with a Reduce task, which typically executes the Reduce function on many key-list-of-values pairs.

proposed algorithm assigns shares appropriately to the values of the non-join attributes of a two-way join for each.

Example: Let $R = (A, B)$ and $S(B, C)$ be two binary relations whose join we want to compute. Suppose $B = 10$ is a heavy hitter. Let R have $r = 1,000 \times 10^9$ tuples with $B = 10$ and S have $s = 250 \times 10^9$ tuples with $B = 10$.[2] Suppose we want to use $k = 400$ reducers. The standard skew-join algorithm partitions the tuples of R with $B = 10$ to the 400 reducers and sends all tuples in S with $B = 10$ to all 400 reducers. Then the communication cost is $r + ks = 1000 \times 10^9 + 400 \times 250 \times 10^9 = 1.01 \times 10^{14}$.

But we can do much better. We identify each of the 400 reducers by a pair (i, j), $i = 1, 2, \ldots, 80$ and $j = 1, 2, 3, 4, 5$. We define a hash function h_r that sends each A value to exactly one out of $i = 80$ buckets and a hash function h_s that similarly partitions C values to $j = 5$ buckets. We still have $80 \times 5 = 400$ reducers. Now we send each tuple $(a, 10)$ of relation R to the following 5 reducers: $(1, h_r(a)), (2, h_r(a)), \ldots, (5, h_r(a))$. Similarly we send each tuple $(10, c)$ of relation S to 80 reducers $(h_s(c), 1), (h_s(c), 2), \ldots (h_s(c), 80)$. Thus we have communication

$$5r + 80s = 5 \times 1,000 \times 10^9 + 80 \times 250 \times 10^9 = 2.5 \times 10^{13}$$

Hence we have reduced the communication by a factor of almost 4. As we will analyze now we can do even better if we chose 10 shares for relation R (instead of 5) and 40 shares for relation S (instead of 80). This will give communication cost equal to 2×10^{13}, which is the best communication cost we can achieve.

7.2 Analysis of Communication Cost

Let $R = (A, B)$ and $S(B, C)$ be two binary relations whose join we want to compute. Suppose $B = b$ is a heavy hitter. Let R have $r > s$ tuples with $B = b$ and S have s tuples with $B = b$.

We partition the tuples of R with $B = b$ into x groups and we also partition the tuples of S with $B = b$ into y groups, where $xy = k$. We use one of the k reducers for each pair (i, j) for a group i from R and for a group j from S. Then we send each tuple (a,b) of R to all reducers of the form (i, q), where $i = h_r(a)$ is the group in which tuple (a, b) belongs and q ranges over all y groups. Similarly for each tuple (b, c) from relation S. Thus each tuple with $B = b$ from R is sent to y reducers and each tuple with $B = b$ from S is sent to x reducers. Hence the

[2]Evidently, it is not feasible to output this join in its entirety, but we can assume that the join is followed by some aggregation, e.g., finding the maximum sum of $A + C$ for any tuple in the join. In Pig or Hive implementation, this aggregation would be done by the same reducers that produce the join, so the join itself would never have to be materialized.

communication cost is $ry + sx$. We will show that by minimizing $ry + sx$ under the constraint $xy = k$ we achieve communication cost equal to $2\sqrt{krs}$.

In order to prove that $2\sqrt{krs}$ is optimal, we use the method of Lagrangean multipliers. We begin with the equation $ry+sx-\lambda(xy-k)$, take partial derivatives with respect to the two variables x and y, and set the results equal to zero. We thus get the following equations:

1. $r = \lambda x$, which implies $ry = \lambda xy = \lambda k$.

2. $s = \lambda y$, which implies $sx = \lambda xy = \lambda k$.

If we multiply (1) and (2) we get $rsxy = rsk = \lambda^2 k^2$, which implies $\lambda = \sqrt{rs/k}$. From (1) we get $x = \sqrt{kr/s}$ and from (2) we get $y = \sqrt{ks/r}$. Thus the communication cost, which is $ry + sx$, is seen equal to $2\sqrt{krs}$.

On the other hand, the standard skew join, assuming $r \geq s$, partitions the tuples of R with $B = b$ among all of the k available reducers and replicates all tuples in S with $B = b$ to all k reducers. Then the communication cost is $r + ks$. If we compare this expression with the optimum that we computed above, $2\sqrt{krs}$, the improvement is significant. The optimal communication cost grows as $\sqrt{k}$, while $r + ks$ grows linearly with k. For example, suppose $s = r$. Then $r + ks = r(k + 1)$, while $2\sqrt{ksr} = 2r\sqrt{k}$.

When there is more than one heavy hitter, then we apply the above technique for each, after we have decided the number of reducers we assign to each.

7.3 Experiments

We have implemented our optimal Algorithm in Java/Hadoop and run experiments on Infolab Hadoop Cluster of Stanford University. This cluster has 21 data nodes with a total of 254 cores, 448 GB of RAM and 18.93 TB of disk space. We suppose some value b for attribute B is identified as a heavy hitter in the join of $R(A, B)$ with $S(B, C)$. In our experiment we have $r = 4 \cdot 10^4$ tuples of R with $B = b$ and we also have $s = 10^4$ tuples of S with $B = b$. The size of the output join for this heavy hitter has $r \cdot s = 4 \cdot 10^8$ tuples and occupies 6.98 GB of disk space. We use $k = 4,000$ buckets. Using our optimal Algrorithm Hadoop computes a total communication cost of $2,500,157$ that matches the theoretical lower-bound of communication which is $2\sqrt{krs} = 2,529,822$. In contrast if we use the standard skew-join algorithm, Hadoop computes a total communication cost of $40,044,001$ (the theoretical cost of standard skew-join algorithm is $r + ks = 40,040,000$). This large difference in the communication cost between the standard and our proposed Algorithm has a direct impact in the total clock time of the computation: Using our Algorithm the total time of the computation is 7 minutes and 40 seconds while when using the standard algorithm for the same computation, the time rises to 10 minutes and 20 seconds. We summarise our experiment results on Figure 7.1.

	Communication cost	Total Time
Standard skew-join Algorithm	$40,044,001$	10 minutes and 20 seconds
The proposed optimal Algorithm	$2,500,157$	7 minutes and 40 seconds

Figure 7.1: Communication cost and time comparison of our experiment when using our optimal proposed Algorithm versus using standard skew-join algorithm for a heavy hitter with $4 \cdot 10^8$ tuples in the join output.

Chapter 8

Conclusions and Open Problems

In this thesis we investigate five problems of databases with uncertainty and lineage. We start with the analysis for those databases of the *query containment* and *query equivalence* problems. We introduce several variants of ULDB database containment and corresponding variants of query containment. We show that the variants of ULDB database containment are all different. However, when we investigate conjunctive query containment we show that the various variants can be partitioned in two classes that are CQ query containment equivalent. We prove that the complexity of *query containment* for databases with uncertainty and lineage for *conjunctive queries (CQs)* and for *Unions of conjunctive queries (UCQs)* remains *NP-complete* as it is with ordinary databases for all five new kinds of containment semantics. We prove that the complexity of *query equivalence* for databases with uncertainty and lineage for CQs and UCQs is also *NP-complete* for the first two kinds of ULDB database containment and *Graph-Isomorphism-complete* for the last three. On another perspective database containment was defined in [ASUW10] for uncertain databases without lineage. We also investigate CQ query containment under this definition. We further extend it for the case of ULDBs by defining and studying the CQ query containment complexity of five new semantics of ULDB equality containment and show that they are useful in ULDB data integration.

Various open problems remain, starting with finding the complexity of ULDB database containment for the various kinds of containment we have defined here. A first open problem is how query optimization techniques can benefit from this investigation of query containment. Another direction is to show that the, related to data exchange, problem of ULDB data integration can benefit from our query containment results. Future investigation can go beyond CQs and UCQs: define a new query language suitable for databases with uncertainty, i.e., that will support uncertainty or lineage querying and study the query containment problem for this uncertain language. Another open area is to study containment and complexity for richer languages, e.g., CQs with negation.

Further we investigate the problem of query answering in a data exchange setting in which the source data that is to be exchanged has uncertainty and lineage. We present and define new logical semantics of *certain answers* for data with uncertainty and lineage. A straightforward way to compute such ULDB certain answers is to first compute all the possible LDB instances of the source and compute LDB certain answers for each one. Even though LDB certain answering of CQs is polynomial for a set of weakly acyclic tgds, the number of the possible instances of a ULDB may be exponential, making this approach computationally expensive and unsuitable for large data sets. In contrast we presented a u-chase procedure that can be used in order to polynomially compute ULDB certain answers of CQs for a set of weakly acyclic tgds and a well-behaved ULDB source. U-chase will create a "pseudo-LDB", use our l-chase procedure on it and finally return a ULDB. Finally we showed that computing certain answers for CQs is no longer polynomial if we allow egds in our dependencies, contrary to what happens in certain data exchange.

We next study and analyse the problem of query computing for conjunctive queries for databases with uncertainty and lineage when we attach belief values to uncertain data that come from a *possibility distribution*. We prove that the model of databases with uncertainty and lineage and with possibilistic values is *closed* for conjunctive queries. This result solves the problem of previous results that show that models with uncertainty and possibilities (but no lineage) are not closed for conjunctive queries [BP05]. We prove that in the model of databases with uncertainty and lineage and possibilities we can compute conjunctive queries (along with their possibility values) with low polynomial complexity (Ptime) in contrast with the high complexity #P of existing approaches where uncertainty values are probabilistic [DS04, GT06, RPT11, STW08].

The fourth problem that we study is the problem of how to *efficiently* compute *aggregate queries* and specifically SUM queries, posed on *big data*. We have presented a method that computes lineage for aggregate queries by applying weighted sampling. The aggregate lineage can be used to compute arbitrary test aggregate queries on subsets of the original data. However the test queries can be computed with good approximation only if the result of each test query is large enough with respect to the total sum over all the data. The aggregate lineage we compute cannot be used to compute test queries if their result is comparatively small. We give performance guarantees. The idea of getting a single (possibly weighted) random sample from a large data set and using it for repeated estimations of a given quantity has appeared before in the context of machine learning and statistical estimation. Boosting techniques [Sch03] such as bootstrapping [Efr79, ET93] are used. In [KTSJ12] BLB is used for the efficient estimation of bootstrap-based quantities in a distributed computational environment.

Finally we study, analyse and implement in the parallel environment *MapReduce* the computation of 2-Way Joins for *big data* that may also be *skewed*. We present a new algorithm, suitable for MapReduce, which computes 2-Way Joins and can efficiently handle *data skewness* in contrast with existing algorithms. We prove that our algorithm matches the lower communication cost bound. We further implement in Java/Hadoop of our algorithm with experiments whose communication cost verifies the theoretical cost.

It was only a few years ago that the optimal serial algorithm to take a multiway join was discovered [NPRR12]. Previous algorithms could be significantly suboptimal when there was significant skew and/or there were many *dangling tuples* (tuples that do not contribute to the result). This result does not extend directly to parallel algorithms. Our result leads to an algorithm that minimizes communication for a MapReduce algorithm, when we constrain the problem by limiting the number of tuples that can be sent to any one reducer. This constraint is reasonable, because controlling the number of inputs to a reducer allows us to guarantee that the Reduce function can be applied in main memory, and it also allows us to force a desired degree of parallelism. These two motivations were discussed in [ASSU13]. There is a suggestion that one can find parallel algorithms for multiway join in a manner that combines techniques such as we have proposed and techniques for optimal serial algorithms [Ré14], but at present no such algorithm is known.

Bibliography

[ABS$^+$12] Foto N. Afrati, Magdalena Balazinska, Anish Das Sarma, Bill Howe, Semih Salihoglu, and Jeffrey D. Ullman. Designing good algorithms for mapreduce and beyond. In *SoCC*, page 26, 2012.

[ADG10] Foto N. Afrati, Matthew Damigos, and Manolis Gergatsoulis. Query containment under bag and bag-set semantics. *Inf. Process. Lett.*, 110(10):360–369, 2010.

[AFV14] Foto N. Afrati, Dimitris Fotakis, and Angelos Vasilakopoulos. Efficient lineage for sum aggregate queries. *To appear in AI Communications Journal*, 2014.

[AK08] Foto N. Afrati and Phokion G. Kolaitis. Answering aggregate queries in data exchange. In *PODS*, pages 129–138, 2008.

[AK10] Foto N. Afrati and Nikos Kiourtis. Computing certain answers in the presence of dependencies. *Inf. Syst.*, 35(2):149–169, 2010.

[AKG91] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, 78(1):158–187, 1991.

[ALM06] Foto N. Afrati, Chen Li, and Prasenjit Mitra. Rewriting queries using views in the presence of arithmetic comparisons. *Theor. Comput. Sci.*, 368(1-2):88–123, 2006.

[ALP08a] Foto Afrati, Chen Li, and Vassia Pavlaki. Data exchange: Query answering for incomplete data sources. In *InfoScale '08: Proceedings of the 3rd international conference on Scalable information systems*, pages 1–10. ICST, 2008.

[ALP08b] Foto N. Afrati, Chen Li, and Vassia Pavlaki. Data exchange in the presence of arithmetic comparisons. In *EDBT*, pages 487–498, 2008.

[Alt94] I. Althöfer. On sparse approximations to randomized strategies and convex combinations. *Linear Algebra and Applications*, 99:339–355, 1994.

[ALU07] Foto N. Afrati, Chen Li, and Jeffrey D. Ullman. Using views to generate efficient evaluation plans for queries. *J. Comput. Syst. Sci.*, 73(5):703–724, 2007.

[ASSU13] Foto N. Afrati, Anish Das Sarma, Semih Salihoglu, and Jeffrey D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *PVLDB*, 6(4):277–288, 2013.

[ASUW10] Parag Agrawal, Anish Das Sarma, Jeffrey D. Ullman, and Jennifer Widom. Foundations of uncertain-data integration. *PVLDB*, 3(1):1080–1090, 2010.

[AU11] Foto N. Afrati and Jeffrey D. Ullman. Optimizing multiway joins in a map-reduce environment. *IEEE Trans. Knowl. Data Eng.*, 23(9):1282–1298, 2011.

[AUV] Foto N. Afrati, Jeffrey D. Ullman, and Angelos Vasilakopoulos. Optimizing 2-way skew joins in mapreduce. *Submitted in International Workshop.*

[AV10a] Foto N. Afrati and Angelos Vasilakopoulos. Managing lineage and uncertainty under a data exchange setting. In *4th International Conference of Scalable Uncertainty Managements (SUM)*, pages 28–41, 2010.

[AV10b] Foto N. Afrati and Angelos Vasilakopoulos. Query containment for databases with uncertainty and lineage. In *4th International Workshop on Management of Uncertain Data (MUD). In conjunction with VLDB 2010.*, pages 67–81, 2010.

[BGMP92] Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 4(5):487–502, 1992.

[BHK98] John S. Breese, David Heckerman, and Carl Myers Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *UAI*, pages 43–52, 1998.

[BKT01] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[BP05] Patrick Bosc and Olivier Pivert. About projection-selection-join queries addressed to possibilistic relational databases. *IEEE T. Fuzzy Systems*, 13(1):124–139, 2005.

[BPP09] Patrick Bosc, Olivier Pivert, and Henri Prade. A model based on possibilistic certainty levels for incomplete databases. In *SUM*, pages 80–94, 2009.

[BSH$^+$08a] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, Martin Theobald, and Jennifer Widom. Databases with uncertainty and lineage. *VLDB J.*, 17(2):243–264, 2008.

[BSH$^+$08b] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, Martin Theobald, and Jennifer Widom. Databases with uncertainty and lineage. *VLDB J.*, 17(2):243–264, 2008.

[BT07] Peter Buneman and Wang Chiew Tan. Provenance in databases. In *SIGMOD Conference*, pages 1171–1173, 2007.

[CGHJ12] Graham Cormode, Minos N. Garofalakis, Peter J. Haas, and Chris Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.

[CGK08] Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *Proceedings of the 21st International Workshop on Description Logics (DL2008), Dresden, Germany, May 13-16, 2008*, 2008.

[CJR08] Adriane Chapman, H. V. Jagadish, and Prakash Ramanan. Efficient provenance storage. In *SIGMOD Conference*, pages 993–1006, 2008.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977.

[CV93] Surajit Chaudhuri and Moshe Y. Vardi. Optimization of *eal* conjunctive queries. In *PODS*, pages 59–70, 1993.

[CW00] Yingwei Cui and Jennifer Widom. Practical lineage tracing in data warehouses. In *ICDE*, pages 367–378, 2000.

[CW03] Yingwei Cui and Jennifer Widom. Lineage tracing for general data warehouse transformations. *VLDB J.*, 12(1):41–58, 2003.

[DFMP04] Didier Dubois, Laurent Foulloy, Gilles Mauris, and Henri Prade. Probability-possibility transformations, triangular fuzzy sets, and probabilistic inequalities. *Reliable Computing*, 10(4):273–297, 2004.

[DGGR02] A. Dobra, M.N. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD Conference*, pages 61–72. ACM, 2002.

[DGGR04] A. Dobra, M.N. Garofalakis, J. Gehrke, and R. Rastogi. Sketch-based multi-query processing over data streams. In *Proc. of the 9th International Conference on Extending Databas Technology (EDBT 2004)*, volume 2992 of *Lecture Notes in Computer Science*, pages 551–568. Springer, 2004.

[DS04] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[Efr79] B. Efron. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

[ES06] Pavlos Efraimidis and Paul G. Spirakis. Weighted random sampling with a reservoir. *Inf. Process. Lett.*, 97(5):181–185, 2006.

[ET93] B. Efron and R. J. Tibshirani. *An Introduction to the Bootstrap.* Chapman & Hall, New York, NY, 1993.

[FKK10] Ronald Fagin, Benny Kimelfeld, and Phokion Kolaitis. Probabilistic data exchange. To appear in 13th International Conference on Database Theory (ICDT), 2010.

[FKMP05] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.

[FKS12] D. Fotakis, A.C. Kaporis, and P.G. Spirakis. Efficient methods for selfish network design. *Theoretical Computer Science*, 448:9–20, 2012.

[GC12] Raman Grover and Michael J. Carey. Extending map-reduce for efficient predicate-based sampling. In *ICDE*, pages 486–497, 2012.

[GKIT07] Todd J. Green, Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Update exchange with mappings and provenance. In *VLDB*, pages 675–686, 2007.

[GKT07] Todd J. Green, Gregory Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[GR02] V. Ganti and R. Ramakrishnan. *Mining and monitoring evolving data*. Springer, 2002.

[Gre09] Todd J. Green. Containment of conjunctive queries on annotated relations. In *ICDT*, pages 296–309, 2009.

[GT06] Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. In *EDBT Workshops*, pages 278–296, 2006.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

[HS07] André Hernich and Nicole Schweikardt. Cwa-solutions for data exchange settings with target dependencies. In *PODS*, pages 113–122, 2007.

[ICF$^+$12] Robert Ikeda, Junsang Cho, Charlie Fang, Semih Salihoglu, Satoshi Torikai, and Jennifer Widom. Provenance-based debugging and drill-down in data-oriented workflows. In *ICDE*, pages 1249–1252, 2012.

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

[IR95] Yannis E. Ioannidis and Raghu Ramakrishnan. Containment of conjunctive queries: Beyond relations as sets. *ACM Trans. Database Syst.*, 20(3):288–324, 1995.

[Jan09] Dietmar Jannach. Fast computation of query relaxations for knowledge-based recommenders. *AI Commun.*, 22(4):235–248, 2009.

[KIT10] Grigoris Karvounarakis, Zachary G. Ives, and Val Tannen. Querying data provenance. In *SIGMOD Conference*, pages 951–962, 2010.

[KTSJ12] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael I. Jordan. The big data bootstrap. In *ICML*, 2012.

[KvdLvW09] Martijn Kagie, Matthijs van der Loos, and Michiel C. van Wezel. Including item characteristics in the probabilistic latent semantic analysis model for collaborative filtering. *AI Commun.*, 22(4):249–265, 2009.

[LLRS97] Laks V. S. Lakshmanan, Nicola Leone, Robert B. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Trans. Database Syst.*, 22(3):419–469, 1997.

[LMM03] R.J. Lipton, E. Markakis, and A. Mehta. Playing large games using simple strategies. In *Proc. of the4th ACM Conference on Electronic Commerce (EC '03)*, pages 36–41, 2003.

[LY94] R.J. Lipton and N.E. Young. Simple strategies for large zero-sum games with applications to complexity theory. In *Proc. of the26th ACM Symposium on Theory of Computing (STOC '94)*, pages 734–740, 1994.

[Mat79] Rudolf Mathon. A note on the graph isomorphism counting problem. *Inf. Process. Lett.*, 8(3):131–132, 1979.

[MBM13] Kivanç Muslu, Yuriy Brun, and Alexandra Meliou. Data debugging with continuous testing. In *ESEC/SIGSOFT FSE*, pages 631–634, 2013.

[MGNS11] Alexandra Meliou, Wolfgang Gatterbauer, Suman Nath, and Dan Suciu. Tracing data errors with view-conditioned causality. In *SIGMOD Conference*, pages 505–516, 2011.

[MM09] Matteo Magnani and Danilo Montesi. Towards relational schema uncertainty. In *SUM*, pages 150–164, 2009.

[NPRR12] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In *PODS*, pages 37–48, 2012.

[ora13a] *Information Management and Big Data.* An Oracle White Paper, February 2013.

[ora13b] *Big Data Analytics - Advanced Analytics in Oracle Database.* An Oracle White Paper, March 2013.

[ORS+08] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *SIGMOD Conference*, pages 1099–1110, 2008.

[Ré14] Christopher Ré. Links between join processing and convex geometry. In *ICDT*, page 2, 2014.

[RPT11] Sudeepa Roy, Vittorio Perduca, and Val Tannen. Faster query answering in probabilistic databases using read-once functions. In *ICDT*, pages 232–243, 2011.

[RRSK11] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011.

[Sch03] R. Schapire. The boosting approach to machine learning: An overview. In *Nonlinear Estimation and Classification*, 2003.

[SDH] A. Das Sarma, L. Dong, and A. Halevy. Uncertainty in data integration. *In "Managing and Mining Uncertain Data" Editor C. Aggarwal, Springer, 2009.*

[STW08] Anish Das Sarma, Martin Theobald, and Jennifer Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *ICDE*, pages 1023–1032, 2008.

[SUW09] Anish Das Sarma, Jeffrey D. Ullman, and Jennifer Widom. Schema design for uncertain databases. In *AMW*, 2009.

[TSJ+09] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive - a warehousing solution over a map-reduce framework. *PVLDB*, 2(2):1626–1629, 2009.

[TSJ+10] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Anthony, Hao Liu, and Raghotham Murthy. Hive - a petabyte scale data warehouse using hadoop. In *ICDE*, pages 996–1005, 2010.

[Ull97] Jeffrey D. Ullman. Information integration using logical views. In *ICDT*, pages 19–40, 1997.

[VK11] Angelos Vasilakopoulos and Verena Kantere. Efficient query computing for uncertain possibilistic databases with provenance. In *3rd USENIX Workshop on the Theory and Practice of Provenance (TaPP)*, 2011.

[WDY93] Joel L. Wolf, Daniel M. Dias, and Philip S. Yu. A parallel sort merge join algorithm for managing data skew. *IEEE Trans. Parallel Distrib. Syst.*, 4(1):70–86, 1993.