DECLARATIVE INCONSISTENCY HANDLING IN RELATIONAL AND SEMI-STRUCTURED DATABASES

by

Sławomir Staworko

A dissertation
submitted to the Faculty of the Graduate School
of State University of New York at Buffalo
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Department of Computer Science and Engineering

May, 2007

Abstract

In many novel database applications inconsistency becomes an important issue that cannot be addressed with simple techniques like data cleaning. For instance, data conflicts can arise during the integration of independent data sources, and the user may not have privileges to resolve the inconsistencies. The framework of *consistent query answers* (CQA) aims to alleviate the impact of inconsistencies on query evaluation by considering all possible (minimal) *repairs* of the original database. The consistent answers are the answers present in *every* repair. Because the repairs are not materialized, this approach does not modify the state of the database i.e., no information is physically removed from the database. In this thesis we advance the research on consistent query answers in several directions.

First, we address the open question of the complexity of computing consistent query answers in the presence of universal constraints. We show that in general the problem is Π_2^P-complete, but we also show that for acyclic sets of full tuple-generating dependencies and denial constraints, the problem is tractable.

Second, we implement a practical system *Hippo* for computing consistent answers to a broad class of queries w.r.t. an acyclic set of full tuple-generating dependencies and denial constraints. The efficiency of this approach, however, suffers from an excessive number of database calls. We devise several optimizations addressing this problem, which make our system capable of handling large databases.

Next, we investigate extending the CQA framework with user-specified preferences on how to resolve conflicts. In the data integration scenario, for instance, the user may possess partial information on the reliability of the data sources. This kind of information can be used to further refine the quality of consistent query answers. We propose a general framework of *preference-based conflict resolution*, identify a set of desired properties, and investigate their computational implications.

Finally, we propose adapting the framework of CQA to *semi-structured databases*. This direction of study is inspired by the observation that many violations of consistency are encountered in the context of XML applications. We propose the framework of *valid query answers*, study its computational implications, and identify tractable cases. We also present an experimental evaluation of our approach.

Acknowledgments

I would like to express deep gratitude to my supervisor, Dr. Jan Chomicki, for his excellent mentorship, continued support, and indispensable guidance in my struggle to get things right.

I would also like to thank Dr. Jerzy Marcinkowski for his unique ability to give hints in doses that help solve the problems without taking away the pleasure of finding the solution.

Last, but not least, my gratitude goes to the many fellow researchers who offered their insightful observations and shared their useful comments.

This research has been supported by NSF Grants IIS-0119186 and IIS-0307434.

To my closest friends:
David Harvey, *Maciek Falski*, and *Xi Zhang*.

Contents

Chapter 1

Introduction

1.1 Motivation

The main role of integrity constraints is to express the semantic properties of data stored in a database. These properties are often actively used to effectively perform operations on the stored data [AHV95, RG00]. Traditionally, the DBMS is responsible for maintaining database consistency with the integrity constraints. Typically, any data manipulation that leads to an integrity violation cannot be performed.

This simple prevention approach, however, is no longer feasible to handle violations of integrity constraints arising in many novel database applications. A typical example is data integration of multiple data sources; even if the individual sources are consistent, together they can contribute conflicting information. At the same time the data sources are often autonomous, and their contents cannot be changed easily or at all.

Example 1.1 Consider the schema

$$Mgr(Name, Dept, Salary)$$

together with with two key dependencies:

$$Name \rightarrow Dept\,Salary, \tag{fd_1}$$

$$Dept \rightarrow Name\,Salary. \tag{fd_2}$$

In an instance of this schema a tuple $Mgr(x, y, z)$ denotes the fact that x manages the department y and receives the salary z. A person can manage only one department and each department can have only one manager.

Now, suppose we integrate the following (consistent) *source* instances:

$$s_1 = \{Mgr(Mary, R\&D, 40k), Mgr(John, PR, 50k), Mgr(Peter, IT, 45k)\}$$

and

$$s_2 = \{Mgr(Mary, PR, 30k), Mgr(John, R\&D, 45k), Mgr(Peter, AD, 35k)\}$$

to obtain the *target* instance $r = s_1 \cup s_2$. We note that this is one of the simplest integration scenarios. Typically, the sources have different schemata that overlap only partially, and the state of the target instance is defined using *mapping rules* [Ull97, Lib06]. The target instance r contains five *conflicts* (violations of functional dependencies):

1. $Mgr(Mary, R\&D, 40k)$ and $Mgr(Mary, PR, 30k)$ w.r.t. fd_1,
2. $Mgr(John, PR, 50k)$ and $Mgr(John, R\&D, 45k)$ w.r.t. fd_1,
3. $Mgr(Peter, IT, 45k)$ and $Mgr(Peter, AD, 35k)$ w.r.t. fd_1,
4. $Mgr(Mary, R\&D, 40k)$ and $(John, R\&D, 45k)$ w.r.t. fd_2,
5. $Mgr(Mary, PR, 30k)$ and $Mgr(John, PR, 50k)$ w.r.t. fd_2.

These conflicts may result from changes that are not yet fully propagated. For instance, $Mary$ may have been promoted to manage $R\&D$, whose previous manager was moved to manage PR, or conversely, $John$ may have been moved to manage $R\&D$, while $Mary$ was moved from $R\&D$ to manage PR. Similarly, $Peter$ may have been moved to manage IT, while previously being the manager of AD, or $Peter$ may have been moved from IT to AD.

Violations of integrity constraints also occur naturally in the context of long and complex data manipulations, delayed updates on data warehouses, and legacy databases. Finally, integrity enforcement may be deactivated for efficiency reasons.

One way of handling inconsistent databases is *data cleaning* [Lom00, DJ03, EBR$^+$01]. Typically, this solution uses conflict resolution rules to specify actions (tuple deletion, tuple insertion, attribute modification, etc.) that need to be performed to restore consistency. Although data cleaning can be very successful, in most cases it is a semi-automatic process requiring a significant amount of user attention. But even then, the user may lack the knowledge or privileges to decide how to resolve every conflict. Also, data cleaning permanently alters the state of the database, and the user may be inclined to consider several possible resolutions rather than committing to a particular one. Finally, occasional violations of integrity constraints may simply indicate exceptions from general rules expressed by the constraints; then, data cleaning leads to information loss.

The semantic properties captured by integrity constraints play a very important role in the way the user formulates a query. If, however, the query is evaluated over an inconsistent database, the results may be wrong and misleading.

Example 1.2 (cont. of Example 1.1) Consider the query Q_1, which asks if *Peter* earns more than *Mary*:

$$Q_1 \equiv \exists d_1, s_1, d_2, s_2. Mgr(Peter, d_1, s_1) \wedge Mgr(Mary, d_2, s_2) \wedge s_1 > s_2.$$

The answer to Q_1 in r is *true*, but this is misleading because r may not correspond to the actual state of the world.

1.2 Consistent query answers

To identify answers that are affected by integrity violations, and therefore may be wrong, Arenas et al. proposed a formal characterization of *consistent query answer* [ABC99]. An answer is consistent if it is present in every possible repair. A *repair* is a consistent instance minimally different from the original one. Intuitively, the answer is consistent if it is obtained regardless of the way the consistency of the database is restored. This framework has served as a foundation for most of the subsequent work in the area of querying inconsistent databases (for the surveys of the area, see [BC03, Ber06, CM05, Cho07]).

Example 1.3 (cont. of Example 1.2) The set of repairs of r consists of four instances:

$$r_1 = \{Mgr(Mary, R\&D, 40k), Mgr(John, PR, 50k), Mgr(Peter, IT, 45k)\},$$

$$r_2 = \{Mgr(Mary, R\&D, 40k), Mgr(John, PR, 50k), Mgr(Peter, AD, 35k)\},$$

$$r_3 = \{Mgr(Mary, PR, 30k), Mgr(John, R\&D, 45k), Mgr(Peter, IT, 45k)\},$$

$$r_4 = \{Mgr(Mary, PR, 30k), Mgr(John, P\&D, 45k), Mgr(Peter, AD, 35k)\}.$$

True is not a consistent answer to Q_1 because Q_1 is false in r_2.

The notion of minimality originally used to define repairs also uses the symmetric set difference. Other investigated notions of minimality use asymmetric set difference [CLR03] and the cardinality of the symmetric difference [ABC03a, LB07] (leading to *Dalal* repairs). Finally, various notions of minimality have been considered to accommodate repairs obtained by attribute value modifications [LB07, BBFL05, Lop06, BFFR05, Wij03].

1.2.1 Computational complexity

It has been observed very early in the research on consistent query answers that the number of possible (minimal) repairs may be exponential [ABC$^+$03b]. A naïve approach to compute consistent query answers by materializing all repairs and consequently evaluating the query in every repair can be highly inefficient. Therefore, the central question of research in the area of consistent query answers is the tractability of the framework. To answer this important question, two fundamental decision problems have been investigated: (1) repair checking – finding if a given database is a repair, and (2) consistent query answering – finding if an answer to a query is present in every repair. Most of the research in the area of consistent query answers uses the notion of data complexity which allows to express the complexity of the problems in terms of the database size only (the set of integrity constraints and the query are assumed to be fixed). The data complexity of computing consistent query answers is:

- PTIME for the class of binary universal constraints and a restricted class of conjunctive queries [ABC99].

- PTIME for the class of denial constraints and quantified-free ground queries [CM05].
- PTIME when at most one primary key per relation is present and the queries belong to a restricted class of conjunctive queries C_{forest} [FM05, Fux07].
- coNP-complete for primary keys and arbitrary conjunctive queries [CM05].
- Π_2^p-complete for arbitrary sets of functional dependencies and inclusion dependencies [CM05]. We note that this problem has been studied for a variety of different semantics of consistent query answers, and we cite the result for only one of them. However, our work does not focus on general inclusion dependencies.

1.2.2 Effective computation of consistent query answers

In general, three different approaches to compute consistent query answers have been proposed: query rewriting, logic programs, and using compact repair representations.

Query rewriting

Query rewriting was the original approach proposed to compute consistent query answers [ABC99]. A query Q is rewritten into a query Q', which upon evaluation returns the set of consistent answers to Q. The main advantages of this approach are the ease of incorporating it into existing database applications and a relatively small overhead in terms of running time. The main shortcoming of this approach is that it cannot be applied to every query: certain classes of conjunctive queries have been shown not to have rewritings [CM05].

To our best knowledge there is only one system implementing the approach of query rewriting to compute consistent answers: ConQuer [FFM05]. This system handles a practical subclass C_{forest} of conjunctive queries in the presence of key dependencies (at most one per relation).

Compact representation of repairs

For certain classes of integrity constraints, it is possible to construct a *compact representation* of all repairs that is polynomial in the size of the database instance. An example of such a construction is the *conflict graph* [CM05]. The vertices of a conflict graph are tuples from the database, and an edge is a set of tuples whose mutual existence in the database violates a denial constraint. Special algorithms have been developed to use conflict graphs to effectively compute consistent query answers to quantifier-free queries [CM05] and scalar aggregation queries [ABC$^+$03b].

Another compact representation of all repairs is *nucleus* [Wij03, Wij05]. In this approach all repairs are represented by a tableau (a table with free variables), and queries are evaluated in the standard way (answers with variables are discarded). We note that for some classes of constraints, constructing the nucleus may take an exponential time to complete.

There is a significant analogy between computing consistent query answers using query rewriting and compact representation of repairs: both approaches try to find if there exists a repair for which the query answer is not present. In query rewriting this is accomplished by executing subqueries constructed from the set of integrity constraints. Because compact representations of repairs contain all the information necessary to find whether it is possible to construct a repair containing some tuple(s), the execution of subqueries can be eliminated. This observation suggests that systems using compact representations of repairs might have better scalability properties.

Logic programs

Several different approaches have been developed to specify all repairs of a database using logic programs with disjunction and classical negation [ABC03a, BB03, EFGL03, GGZ01, GGZ03, VNV02]. Usually, every (stable) model corresponds to a repair, and consistent query answers to a query can be obtained by using reasoning under cautious semantics. Such programs can then be evaluated using an existing logic programming system like `dlv` [EFLP00]. These approaches have the advantage of generality, as typically arbitrary first-order queries and universal constraints (or even some referential integrity constraints [ABC03a]) can be handled. However, the generality comes at a price: reasoning under the cautious semantics for the classes of logic programs used is known to be Π_2^p-complete. Another shortcoming of this approach is that systems like `dlv` perform *grounding* of the specified program, which limits its practical uses for large databases.

To address the described difficulties, the INFOMIX system [EFGL03] uses several optimizations geared toward efficient evaluation of logic programs computing consistent query answers. One is localization of conflict resolution. Another is encoding tuple membership in individual repairs using bit-vectors, which makes possible efficient computation of consistent query answers using bit-wise operators. However, it is known that even in the presence of one functional dependency there may be *exponentially* many repairs. With only 80 tuples involved in conflicts, the number of repairs may exceed 10^{12}! It may be impractical to efficiently manipulate bit-vectors of that size, which may limit scalability of this approach. Very recently, [EFGL07] proposed to address this deficiency with *repair factorization*.

1.3 Our contributions

In this document we describe our research in several directions relevant to consistent query answers.

Universal constraints

Denial constraints allow the user to specify sets of tuples that cannot be all present in the database at the same time: those tuples create a conflict. Their simultaneous presence in the database is recorded in the conflict graph which is used to efficiently compute consistent query answers [CM05].

At the same time, Arenas et al. in [ABC99] present a rewriting technique for computing consistent query answers to binary universal constraints. Universal constraints allow to express conflicts created not only by the presence of some tuples but also by a simultaneous absence of other tuples.

In Chapter 3 we address the open question of the complexity of computing consistent query answers in the presence of universal constraints. We investigate an extended version of conflict graph, which captures conflicts w.r.t. universal constraints. Although, the extended conflict graph does not have exactly the same properties as its basic form, we use it to compute consistent query answers in polynomial time in the presence of denial constraints and acyclic full tuple-generating dependencies. We also show that dropping the restriction of acyclicty leads to coNP-completeness and that computing consistent query answers in the presence of arbitrary universal constraints is Π_2^p-complete.

System Hippo

Despite a considerable amount of research devoted to the study of computational complexity of consistent query answers, there has been relatively limited effort in investigating practical aspects of computing consistent query answers. In Chapter 4 we pursue this interesting challenge and construct a practical and efficient system Hippo. As theoretical foundations we use the results from Chapter 3 and [CM05]. Hippo computes consistent answers to projection-free queries in the presence of denial constraints and acyclic full tuple-generating dependencies. Thanks to various optimizations, the system is very efficient: virtually no overhead is needed to compute consistent answers to Select-Join queries, and queries with union take about twice the time of evaluating the query against the database.

Preference-driven conflict resolution

In many applications, the user has a preference on how some conflicts should be resolved. Typical information used to express the preference includes:

- the timestamp of creation/last modification of the tuple: the conflicts can be resolved by removing from consideration old, outdated tuples;
- the source of the information of the tuple: the user can consider the data from one source to be more reliable than the data from the other;
- and some attributes of the conflicting tuples.

Example 1.4 (cont. of Example 1.3) Suppose that for a conflict created by two tuples referring to the same person (a violation of fd_1), the user prefers to resolve the conflict by removing the tuple with the smaller salary. This preference expresses the belief that if a manager is being reassigned, her salary is not lowered. In r this preference applies to Conflict 3: the tuple $Mgr(Peter, IT, 45k)$ is preferred over $Mgr(Peter, AD, 30k)$ as the resolution of Conflict 3. This preference also applies to Conflicts 1 and 2. It doesn't apply, however, to Conflicts 4 and 5 because both of them involve tuples referring to different persons.

Unfortunately, the standard framework of consistent query answers does not provide a way to include user preferences. To address this deficiency, in Chapter 5 we extend the framework of consistent query answers to use preference information. We define a set of *preferred repairs* (a subset of all repairs.) Query answers present in every preferred repair are called *preferred consistent query answers.* Naturally, there may be more than one way to select the preferred repairs. We investigate three different ones.

In Example 1.4 we eliminate from considerations the repairs r_2 and r_4 because the preference information applies to Conflict 3. Similarly, by applying the preference information to Conflicts 1 and 2, we may also eliminate the repair r_3. Then, the repair r_1 is the only preferred repair and *true* is the preferred consistent answer to Q_1. This way of selecting preferred repairs is captured by the notion of *global optimality* of a repair. In this example global optimality of a repair coincides with another investigated notion of a *common* repair, which uses the preference information to guide a procedural approach for constructing a preferred repair.

However, one may find the reasons for eliminating repair r_3 insufficient: there is no preference information that applies to Conflicts 4 and 5, yet not all of their possible resolutions are considered. In that case the set of preferred repairs consists of r_1 and r_3. This way of selecting preferred repairs is captured by a weaker notion of *Pareto optimality.*

Consistent query answers for XML databases

XML is rapidly becoming the standard format for the representation and exchange of semi-structured data over the Internet. In most contexts, documents are processed in the presence of a schema, typically a Document Type Definition (DTD) or an XML Schema. Although currently there exist various methods for maintaining the validity of semi-structured databases, many XML-based applications operate on data that is invalid with respect to a given schema. A document may be the result of an integration of several documents of which some are not valid. Parts of an XML document could be imported from a document that is valid w.r.t. a schema slightly different than the given one. For example, the schemata may differ with respect to the constraints on the cardinalities of elements. The presence of legacy data sources may even result in situations where schema constraints cannot be enforced at all. Also, violations of the schema are often caused by manual data entry.

At the same time, DTDs and XML Schemas capture important information about the expected structure of an XML document. This information plays a crucial role when defining queries and transformations of the documents. Similar to the relational case, executing queries and transformations on invalid documents may lead to results that are insufficiently informative or do not conform to the expectations of the user.

In Chapter 6 we investigate the problems of repairing and querying invalid XML documents. We adopt the notion of repair and consistent query answers to semi-structured databases. In this scenario we capture the differences between documents using sequences of editing operations: inserting a subtree, deleting a subtree, and modifying the label of a node. Such operations are commonly used in the context of XML document change detection [CRGMW96, CAM02] and incremental integrity maintenance of XML documents [AMR+98, BPV04, BFG05].

We present an efficient algorithm for construction of the *trace graph*, which is a compact representation of all repairs of an invalid XML document. We investigate using this structure to compute edit distance between the document and the DTD and to evaluate XPath queries in a validity-sensitive fashion.

Chapter 2

Preliminaries

In this chapter we review the standard notions of relational databases [AHV95] and present the standard framework of consistent query answers [ABC99] for relational databases.

2.1 Relational model

A database schema $\mathcal{S}$ is a set of relation names of fixed arity. Relation attributes are drawn from an infinite set of names U. We use $A, B, C, \ldots$ to denote elements of U and $X, Y, Z, \ldots$ to denote finite subsets of U.

Every element of U is typed and we consider only two disjoint domains: $\mathbb{Q}$ (decimals) and D (uninterpreted constants). We assume that two constants are equal if and only if they have the same name. We also use the natural interpretation of the *built-in* relation symbols $=$, $\neq$, $<$, $\leq$, $>$, $\geq$ over $\mathbb{Q}$.

Formally, a *tuple* is a (partial) function which takes an attribute and returns a value from the attribute domain. We use $t, t', t'', s, \ldots$ to denote tuples. By $t[A]$ we denote the value of t on the attribute A. Typically, however, we assume an order among the attributes for which a tuple t is defined and then we use subsequent natural numbers $1, \ldots, k$ to refer to the attributes. This allows us to view the tuple t as a finite sequence of values: $(t[1], t[2], \ldots, t[k]) \equiv (t_1, \ldots, t_k)$. Also, $t[i, \ldots, j]$ stands for the tuple consisting of values of t on attributes $i, \ldots, j$, i.e. the tuple $(t_i, \ldots, t_i)$.

Because we deal with schemata with several relation names, when describing the contents of a database instance we additionally assign to a tuple the relation name the tuple belongs to. In other words, when describing tuples of a database we equate the notion of a tuple with the notion of a positive ground fact.

A *database instance* (or simply an *instance*) is a finite set of tuples (together with relation names). We use $I, I', I'', J, \ldots$ to denote instances. We also view the instance I as its *standard model*: a first-order structure over the vocabulary consisting of $\mathcal{S}$ and the built-in binary relation symbols in which the set of true ground facts is equal to I.

2.1.1 Query languages

Throughout this document we use two languages to query relational database instances: relational algebra and first-order logic.

We consider the class of *first-order logic* (FOL) queries defined as follows:

$$\varphi ::= R(\bar{x}) \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \exists x.\varphi,$$

where R is a relation name of $\mathcal{S}$ and $\bar{x}$ is a compatible vector of variables and constants, $\wedge$, $\neg$, $\exists$ are the disjunction operator, the negation operator, and the existential quantifier respectively. We also use the standard macros: $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, and $\forall x.\varphi \equiv \neg\exists x.\neg\varphi$. We define the set of free-variables of φ in the standard way and φ is *closed* if and only if φ has no free variables; otherwise the query is *open* and we explicitly identify its free variables: $\varphi(\bar{x})$. *True* is the *answer* to a closed FOL query φ if and only if $I \models \varphi$ (in the standard model-theoretic way). The set of answers to an open FOL query $\varphi(\bar{x})$ in I is the set of all tuples compatible with $\bar{x}$ which after substitution make the formula *true* in I, i.e.:

$$QA(\varphi(\bar{x}), I) = \{t | I \models \varphi[t]\}.$$

The class of *relational algebra* (RA) expressions we consider is defined by the following grammar:

$$E ::= R \mid \sigma_{\vartheta}(E) \mid \pi_X(E) \mid E_1 \times E_2 \mid E_1 \cup E_2 \mid E_1 \setminus E_2,$$

where $R \in \mathcal{S}$ is any relation name, $\sigma, \pi, \times, \cup, \setminus$ are the selection, projection, Cartesian product, union, and set difference operators respectively. In the selection condition ϑ and the attribute list X, we identify attributes with their position (a natural number). For an relational expression E, by $|E|$ we denote its arity. A tuple t is *compatible* with an expression E if the arities and corresponding types of t and E agree.

We evaluate an RA expression E over I in the standard way and we denote the results by $QA(I, E)$. We recall that for every RA expression there exists an equivalent first-order logic formula and vice versa. Moreover, for any π*-free* RA expression there exists an equivalent quantifier-free FOL query (the converse, however, is not necessarily true).

We also construct extended relational algebra expressions using *left outer join* operator $E \overset{\vartheta}{\leftarrow} E'$, where ϑ is the join condition. We use $E.i$ and $E'.j$ in ϑ to identify the attributes of the expressions E and E'. We denote the *null* value by $\perp$. We also note that we do not consider database instances with null values, and therefore the null values can only be present in the result of an expression with an outer join.

2.1.2 Integrity constraints

In general, an *integrity constraint* is any closed first-order formula over the vocabulary $\mathcal{S}$ and the set of built-in binary relation symbols. Given a set of constraints F, a database I is *consistent* w.r.t. F if and only if $I \models F$; otherwise I is *inconsistent.*

In this document we investigate constraints belonging to the class of *universal constraints* of the following form:

$$\forall \bar{x} \big[\neg R_{i_1}(\bar{x}_1) \vee \ldots \vee \neg R_{i_n}(\bar{x}_n) \vee \neg \rho \vee P_{j_1}(\bar{y}_1) \vee \ldots \vee P_{j_m}(\bar{y}_m) \big]. \tag{2.1}$$

where $\bar{x}_i$ and $\bar{y}_i$ are vectors of variables such that $\bar{y}_1 \cup \bar{y}_2 \cup \ldots \cup \bar{y}_m \subseteq \bar{x}_1 \cup \bar{x}_2 \cup \ldots \cup \bar{x}_n = \bar{x}$ and ρ is a boolean formula using only variables $\bar{x}_i$ and $\bar{y}_i$, constants, and the standard built-it operators. We also denote a universal constraints of the form 2.1 with an implication:

$$R_{i_1}(\bar{x}_1) \wedge \ldots \wedge R_{i_n}(\bar{x}_n) \wedge \rho(\bar{x}) \rightarrow P_{j_1}(\bar{y}_1) \vee \ldots \vee P_{j_m}(\bar{y}_m). \tag{2.2}$$

Also, We call the relation names $P_{j_1}, \ldots, P_{j_m}$ the *head* of the constraint and the relation names $R_{i_1}, \ldots, R_{i_n}$ the *body* of the constraint.

We observe that the class of universal constraints contains the following classes of integrity constraints:

1. *full tuple-generating dependencies* (full TGDs) when $m = 1$;
2. *denial constraints* when $m = 0$;
3. *functional dependencies* (FDs). An FD denoted as $R : X \to Y$ stands for the following formula:

$$\forall \bar{x}, \bar{y}. R(\bar{x}) \wedge R(\bar{y}) \wedge \bigwedge_{A \in X} \bar{x}[A] = \bar{y}[A] \wedge \bigvee_{B \in Y} \bar{x}[B] \neq \bar{y}[B] \to false.$$

Constraint grounding

Given a database instance I and a set of integrity constraints F, the *active domain* of I and F, denoted by $adom(I, F)$, is the set of all constants used in I and F. Note that because we consider only finite database instance and finite sets of integrity constraints, the active domain is also a finite set.

Definition 2.1 (Ground rule) For a universal constraint of the form (2.2), the implication $t_1 \wedge \ldots \wedge t_n \to s_1 \vee \ldots \vee s_m$ is a *ground rule* of the constraint if there exists a (total) substitution substitution[1] θ of the variables $\bar{x}$ such that $R(\bar{x}_i)\theta = t_i$ for $i \in \{1, \ldots, n\}$, $R(\bar{y}_j)\theta = s_j$ for $j \in \{1, \ldots, m\}$, and $\rho\theta$ holds. By $ground(I, F)$ we denote the set of all ground rules obtained from the constraints in F with substitutions that assign values in $adom(I, F)$.

Also, for a ground rule $t_1 \wedge \ldots \wedge t_n \to s_1 \vee \ldots \vee s_m$ we call the set of tuples $\{t_1, \ldots, t_n\}$ a *premise* of s_j for every $j \in \{1, \ldots, m\}$.

Acyclic constraints

Definition 2.2 (Acyclic constraints) The *dependency graph* $\mathcal{D}(F)$ of a set of universal constraints F is a directed graph whose set of vertices is $\mathcal{S}$ and there is an edge coming

[1]We use the postfix notation $f\theta$ to denote substituting the term f with θ.

from R to P if there exists a constraint in F having R in its body and P in its head. F is *acyclic* if and only if $\mathcal{D}(F)$ is acyclic.

We also note that if F is acyclic, then the length of a directed path in $\mathcal{D}(F)$ is bounded by the size of $\mathcal{S}$.

Immediate consequence operator

Now, we consider only sets integrity constraints containing full TGDs and denial constraints only. We observe that because a full TGD contains only one relation name in the head, it explicitly indicates the tuple that must be present in the instance if all the tuples from its body are present and the condition ρ in the body is satisfied. To identify the instance after adding potentially missing tuples we use standard Datalog constructs [AHV95, Bar03].

Definition 2.3 (Immediate consequence operator and its fix point) For a set of full TGDs and denial constraints F and a set of tuples J, the *immediate consequence* of F in J is defined as

$$T_F(J) = J \cup \{s \mid t_1 \land \ldots \land t_n \to s \in \mathit{ground}(I, F) \text{ and } \{t_1, \ldots, t_n\} \subseteq J\}.$$

The *closure* or the *fix-point* of the immediate consequence operator, denoted by T_F^*, is defined in the standard way.

Given an instance I, a tuple is *base* if $t \in I$; otherwise t is *non-base.*

2.1.3 Repairs

For two databases I_1, I_2 their *(symmetric) difference* $\Delta(I_1, I_2)$ is

$$\Delta(I_1, I_2) = I_1 \setminus I_2 \cup I_2 \setminus I_1.$$

Given an instance I we define the *relative proximity relation* $\leq_I$ on instances as follows:

$$I_1 \leq_I I_2 \iff \Delta(I, I_1) \subseteq \Delta(I, I_2).$$

We note that for any I, the relation $\leq_I$ is a partial ordering of all database instances. We use $I_1 <_I I_2$ to denote $I_1 \neq I_2$ and $I_1 \leq_I I_2$.

Definition 2.4 (Repair) Given an instance I and a set of universal constraints F, an instance I' is a *repair* of I w.r.t. F if and only if I' is a $<_I$-minimal instance consistent with F. By $Rep(I, F)$ we denote the set of all repairs of I w.r.t. F.

We note that even if we restrict the set of constraints to one functional dependency, the number of repairs can be exponential in the number of tuples in the database [ABC+03b].

Example 2.5 Consider schema consisting of one relation name $R(A, B)$ and an FD $R : A \rightarrow B$. For any $n \in \mathbb{N}$ the instance

$$I_n = \{R(1,0), R(1,1), \ldots, R(n,0), R(n,1)\}$$

has 2^n repairs.

Definition 2.6 (Consistent answers to closed FOL queries) Given an instance I, a set of universal constraints, and a closed FOL query Q, *true* is the *consistent answer* to Q in I w.r.t. F, denoted by $I \models_F Q$, if and only if *true* is an answer to Q in every repair of I w.r.t. F.

Definition 2.7 (Consistent answers to RA expressions) Given an instance I, a set of universal constraints, and an RA expression E, a tuple t is a *consistent answer* to E in I w.r.t. F if and only if $t \in QA(I', E)$ for every $I' \in Rep(I, F)$. We denote the set of all consistent answers to E by $CQA(I, E, F)$.

2.1.4 Conflicts and conflict hypergraphs

We propose the notion of a conflict to identify constraints violations.

Definition 2.8 (Conflict) For a constraint of the form (2.2) and any substitution θ of the universal variables such that $\rho\theta$ holds, the set of ground facts

$$\{R_{i_1}(\bar{x}_1)\theta, \ldots, R_{i_n}(\bar{x}_n)\theta, \neg P_{j_1}(\bar{y}_1)\theta, \ldots, \neg P_{j_m}(\bar{y}_m)\theta\}$$

is a *conflict* w.r.t. (2.1).

A conflict is a *denial* conflict if it contains no negative facts. Naturally, denial conflicts result only from violations of denial constraints. For denial conflicts we recall the notion of conflict hypergraph that represents all repairs of an instance.

Definition 2.9 (Conflict hypergraph [ABC+03b, CM05]) Given a database instance I and a set of denial constraints F, the *conflict hypergraph* $G(I, F)$ is a hypergraph whose set of vertices is I and the set of hyperedges consists of all conflicts among the elements of I.

If we restrict the integrity constraints to FDs, every conflict is binary and we obtain a standard graph (whose edges connect exactly two vertices).

Example 2.10 The conflict graph for the instance r of Example 1.1 is in Figure 2.1.

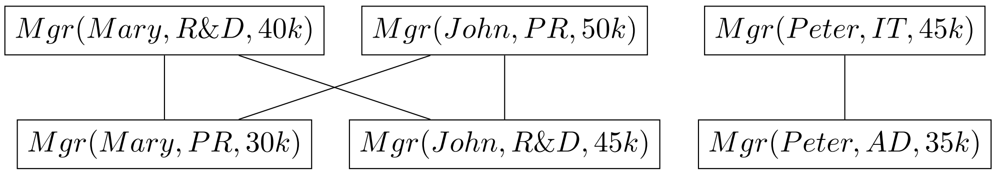


Figure 2.1: A conflict graph.

In the original framework of [ABC99], two operations, inserting a tuple and deleting a tuple, are considered when *repairing* an inconsistent instance. In the presence of denial constraints adding new tuples cannot remove any conflicts, and therefore the repairs are obtained by removing tuples only.

Fact 2.11 *A* maximal independent set *of a conflict hypergraph $G(I, F)$ is any maximal set of vertices that contains no hyperedge. Any maximal independent set of $G(I, F)$ is a repair of I w.r.t. F and vice versa.*

In Chapter 3 we show how to extend the conflict hypergraph to capture repairs w.r.t. universal constraints.

2.1.5 Data complexity

To investigate tractability of the framework of consistent query answers (and its extensions) we use the notion of data complexity [Var82]. When using this notion we describe the complexity of the problems in terms of the size of the database only and assume other parts of the input, like the set of integrity constraints or the query, to be fixed. There are two classical decision problems that are investigated in the context of consistent query answers [CM05]:

(i) *repair checking* – determining if a database instance is a repair of a given database instance, i.e. the complexity of the following set

$$\mathcal{B}_F = \{(I, I') : I' \in Rep(I, F)\}.$$

(ii) *consistent query answering* – determining if *true* the consistent answer to a given closed FOL query in a given database w.r.t. to a given set of integrity constraints, i.e. the complexity of the following set

$$\mathcal{D}_{F,Q} = \{I : I \models_F Q\}.$$

Chapter 3

Universal constraints

In this chapter we address the question of tractability of consistent query answer in the presence of universal constraints.

3.1 Extended conflict hypergraph

We recall that when considering repairs in the presence of denial constraints, the repairs are obtained by deleting tuples only, and hence every repair is a subset of the original database. In the case of universal constraints a violation can be repaired not only by deletion but also by insertion of a tuple. We note that in the case of violations of universal constraints the tuples that can be inserted to resolve the conflict are uniquely identified.

Definition 3.1 (Hull) Given a database instance I and a set of universal constraints F, the *hull* $H(I, F)$ of I w.r.t. F is the minimal set satisfying the following conditions:

1. $I \subseteq H(I, F)$,
2. if a set of facts e is a conflict w.r.t. F and every positive fact of e belongs to $H(I, F)$, then for every negative fact $\neg t$ in e, both $\neg t$ and t belong to $H(I, F)$.

We extend the notion of the conflict hypergraph [ABC$^+$03b, CM05] which represents all repairs using space polynomial in the size of the database. Because conflicts w.r.t. universal constraints involve also negative facts, to use the notion of independent set to capture consistent instances, we need to introduce also edges between any pair of t and $\neg t$. This way an independent set of the extended conflict graph does not contain contradictory facts.

Definition 3.2 (Extended conflict hypergraph) The *extended conflict hypergraph* of I w.r.t. F is $G(I,F) = (H(I,F), E(I,F))$, where $E(I,F)$ contains two types of hyperedges:

1. *conflict hyperedges* $e \subseteq H(I,F)$ such that e is a conflict w.r.t. F,
2. *stabilizing edges* $\{t, \neg t\}$ if both t and $\neg t$ belong to $H(I,F)$.

An *independent set* of an extended conflict hypergraph $G(I,F)$ is any subset of $H(I,F)$ that contains no hyperedges.

We also note that the conflict hyperedges can be viewed as a set of rules obtained from grounding the constraints from F with tuples of $H(I,F)$, i.e. $t_1 \wedge \ldots \wedge t_n \rightarrow s_1 \vee \ldots \vee s_m$ corresponds to $\{t_1, \ldots, t_n, \neg s_1, \ldots, \neg s_m\}$.

For a given set of universal constraints F the definition of the hull yields a simple negation-free Datalog program which generates the set of all facts in the hull. Therefore,

Fact 3.3 *For every F and every I the extended conflict hypergraph $G(I,F)$ can be constructed in time polynomial in the size of I and using space polynomial in the size of I.*

For a set of (positive and negative) facts J its *positive projection*, denoted by J^+, is the set of all positive facts in J (basically a database instance containing all positive facts of J). The correspondence between independent sets of the extended conflict hypergraph and repairs is stated by the following lemma.

Lemma 3.4 *Any repair of I is an independent set of the extended conflict hypergraph. Moreover, for any independent set M of $G(I,F)$ either M^+ is a repair or there exists an independent set N such that $N^+ <_I M^+$.*

Proof

(i) Suppose there exists a repair $I' \in Rep(I,F)$ such that I' is not an independent set of $G(I,F)$. Because I' is consistent, I' contains no edges in $G(I,F)$. Therefore, $I' \not\subseteq H(I,F)$. Take the set $I'' = H(I,F) \cap I'$. We claim that I'' is consistent;

otherwise I' is not consistent or $H(I,F)$ is not the hull. Now, because $I \subseteq H(I,F)$ we have that $I'' <_I I'$ which contradicts I' being a repair.

(ii) Take any independent set M such that M^+ is not a repair. Note that M^+ is consistent and therefore there exists a repair I' such that $I' <_I M^+$. By (i) I' is an independent set of $G(I,F)$.

□

Note: Because all repairs consist of tuples contained in $H(I,F)$, from now on, if we quantify over tuples (or sets of tuples) we mean only tuples in $H(I,F)^+$.

3.2 Complexity of universal constraints

Theorem 3.5 *Consistent query answering for universal constraints is Π_2^p-complete. The reduction uses an atomic query and a set of 3 integrity constraints: one functional dependency, one denial constraint with two atoms, and one universal constraint with one atom in its body and 2 atoms in its head.*

Proof The membership of $\mathcal{D}_{F,Q}$ to Π_2^p follows from the definition of consistent query answers: query is not consistently *true* if and only if it is *false* in some repair; each repair is a subset of the hull of the given instance (and thus of a polynomial size) and checking if a set of tuples is a repair is in coNP (see the proof of Corollary 3.6). We show Π_2^p-hardness below.

Consider the following $\forall^*\exists^*\text{QBF}_3$ formula:

$$\psi = \forall x_1, \dots, x_n . \exists x_{n+1}, \dots, x_{n+m} . \varphi,$$

where $\varphi = c_1 \wedge \ldots \wedge c_k$ is quantifier-free 3CNF i.e., c_j are clauses of three literals $l_{j,1} \vee l_{j,2} \vee l_{j,3}$. For ease of reference, we call the variables $x_1, \dots, x_n$ *universal* and the variables $x_{n+1}, \dots, x_{n+m}$ *existential.* Checking satisfiability of such formulas is a classical Π_2^p-complete problem [Pap94].

We construct the database instance I_ψ over the schema consisting of only one relation name: $R(A, B, V_0, S_0, A_1, B_1, V_1, S_1, A_2, B_2, V_2, S_2, A_3, B_3, V_3, S_3, V_3', S_3')$. The set of integrity constraints F consists of:

$$R : A \to B, \tag{3.1}$$

$$R(A, B, 0, \ldots, 0) \wedge R(C_1, C_2, 0, \ldots, 0) \wedge C_2 > 0 \wedge B > C_2 \to false, \tag{3.2}$$

$$\begin{aligned} R(A, B, &V_0, S_0, A_1, B_1, V_1, S_1, A_2, B_2, V_2, S_2, A_3, B_3, V_3, S_3, V_3', S_3') \to \\ &R(A_1, B_1, V_1, S_1, A_2, B_2, V_2, S_2, A_3, B_3, V_3, S_3, V_3', S_3', 0, 0, 0, 0) \vee \\ &R(V_0, S_0, 0, \ldots, 0). \end{aligned} \tag{3.3}$$

We use the following two auxiliary functions on literals of φ:

$$var(x_i) = i \qquad sgn(x_i) = 0$$
$$var(\neg x_i) = i \qquad sgn(\neg x_i) = -1.$$

To construct the database instance we use the following types of tuples:

- a *dummy* tuple
$$\mathbb{O} = R(0, \ldots, 0);$$
- a *special* tuple
$$s = R(n + m + 1, n, 0, \ldots, 0);$$
- v_i- and $\bar{v}_i$-tuples corresponding resp. to the positive and negative valuation of the variable x_i (for $i \in \{1, \ldots, n + m\}$)
$$v_i = R(i, 0, 0, \ldots, 0), \quad \bar{v}_i = R(i, -1, 0, \ldots, 0);$$

- d_j-tuples corresponding to the conjunct c_j (for $j \in \{1, \dots, k\}$)

$$\begin{aligned} d_j = R(&-4i, 0, n+m+1, n,\\ &3-4i, 0, var(l_{j,3}), sgn(l_{j,3}),\\ &2-4i, 0, var(l_{j,2}), sgn(l_{j,2}),\\ &1-4i, 0, var(l_{j,1}), sgn(l_{j,1}), var(l_{j,1}), sgn(l_{j,1})); \end{aligned}$$

- d_j^3-tuples corresponding to the disjunction of the three literals of conjunct c_j (for $j \in \{1, \dots, k\}$)

$$\begin{aligned} d_j^3 = R(&3-4j, 0, var(l_{j,3}), sgn(l_{j,3}),\\ &2-4j, 0, var(l_{j,2}), sgn(l_{j,2}),\\ &1-4j, 0, var(l_{j,1}), sgn(l_{j,1}),\\ &var(l_{j,1}), sgn(l_{j,1}), 0, 0, 0, 0); \end{aligned}$$

- d_j^2-tuples corresponding to the disjunction of the first two literals of conjunct c_j (for $j \in \{1, \dots, k\}$)

$$\begin{aligned} d_j^2 = R(&2-4j, 0, var(l_{j,2}), sgn(l_{j,2}),\\ &1-4j, 0, var(l_{j,1}), sgn(l_{j,1}),\\ &var(l_{j,1}), sgn(l_{j,1}), 0, 0,\\ &0, 0, 0, 0, 0, 0); \end{aligned}$$

- d_j^1-tuples corresponding to (the disjunction of) the first literal of conjunct c_j (for $j \in \{1, \dots, k\}$)

$$\begin{aligned} d_j^1 = R(&1-4j, 0, var(l_{j,1}), sgn(l_{j,1}),\\ &var(l_{j,1}), sgn(l_{j,1}), 0, 0,\\ &0, 0, 0, 0,\\ &0, 0, 0, 0, 0, 0). \end{aligned}$$

We observe that the type of a tuple is determined by its values on the first two attributes and hence now two different tuples can be confused.

For the clarity of further considerations by $L_{j,p}$ we denote the tuple corresponding to the satisfying valuation of the literal $l_{j,p}$, i.e.

$$L_{j,p} = \begin{cases} v_i & \text{when } l_{j,p} = x_i, \\ \bar{v}_i & \text{when } l_{j,p} = \neg x_i. \end{cases}$$

Also, to remove unnecessary technical details we instantiate the integrity constraints to the following set of implications:

$$v_i \land \bar{v}_i \to false \qquad \text{for } i \in \{1, \ldots, n+m\}, \tag{H1}$$

$$v_i \land s \to false \quad \text{and} \quad \bar{v}_i \land s \to false \qquad \text{for } i \in \{n+1, \ldots, n+m\}, \tag{H2}$$

$$v_i \to \mathbb{O} \quad \text{and} \quad \bar{v}_i \to \mathbb{O} \qquad \text{for } i \in \{1, \ldots, n+k\}, \tag{H3}$$

$$s \to \mathbb{O}, \tag{H4}$$

$$d_j^1 \to L_{j,1} \qquad \text{for } j \in \{1, \ldots, k\}, \tag{H5}$$

$$d_j^2 \to d_j^1 \lor L_{j,2} \qquad \text{for } j \in \{1, \ldots, k\}, \tag{H6}$$

$$d_j^3 \to d_j^2 \lor L_{j,3} \qquad \text{for } j \in \{1, \ldots, k\}, \tag{H7}$$

$$d_j \to d_j^3 \lor s \qquad \text{for } j \in \{1, \ldots, k\}, \tag{H8}$$

The rules (H1) are obtained from grounding the constraint (3.1) and the rules (H2) are obtained from grounding the constraint (3.2). The remaining rules are obtained from grounding the constraint (3.3).

For the formula ψ we construct the following instance:

$$I_\psi = \{\mathbb{O}, v_1, \bar{v}_1, \ldots, v_{n+m}, \bar{v}_{n+m}, d_1^1, d_1^2, d_1^3, \ldots, d_k^1, d_k^2, d_k^3, d_1, \ldots, d_k\}$$

Figure 3.1 contains the extended conflict hypergraph for an instance obtained from the formula $\forall x_1, x_2, x_3. \exists x_4, x_5. (\neg x_1 \lor x_4 \lor x_2) \land (x_3 \lor \neg x_5 \lor \neg x_2)$.

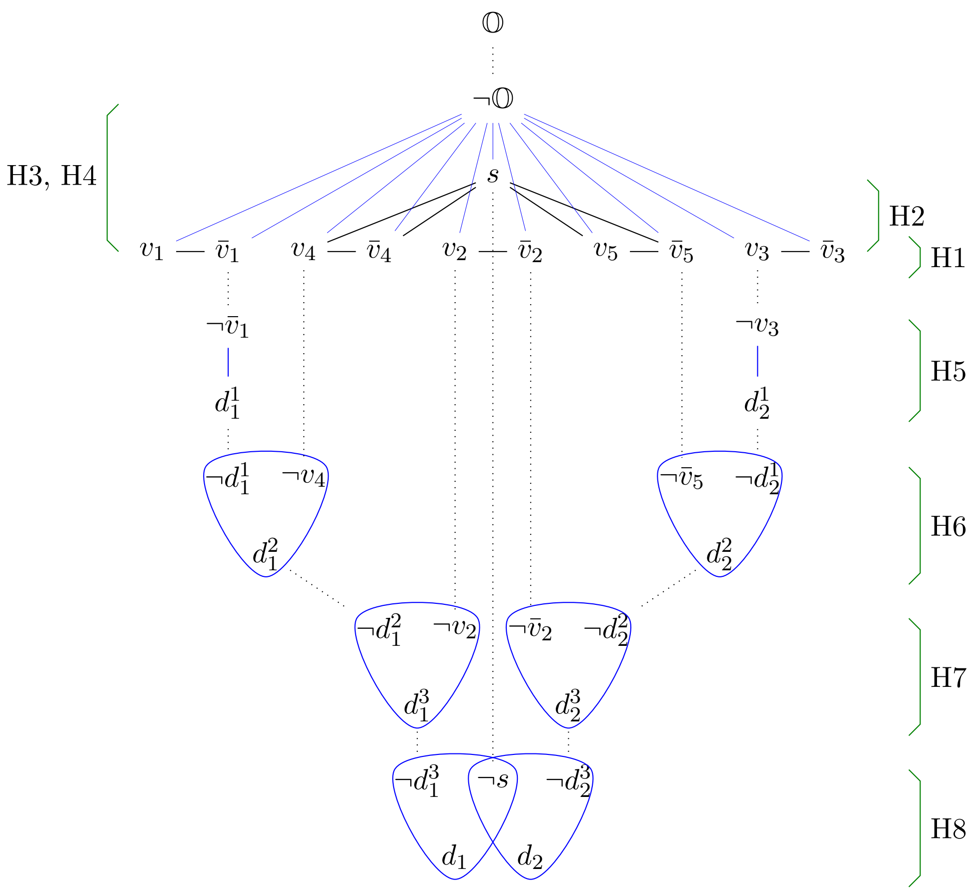

$\mathbb{O}$
$\neg\mathbb{O}$
H3, H4
s
H2
H1
v_1 — $\bar{v}_1$ v_4 — $\bar{v}_4$ v_2 — $\bar{v}_2$ v_5 — $\bar{v}_5$ v_3 — $\bar{v}_3$
$\neg\bar{v}_1$
$\neg v_3$
H5
d_1^1
d_2^1
$\neg d_1^1$ $\neg v_4$ d_1^2
$\neg\bar{v}_5$ $\neg d_2^1$ d_2^2
H6
$\neg d_1^2$ $\neg v_2$ d_1^3
$\neg\bar{v}_2$ $\neg d_2^2$ d_2^3
H7
$\neg d_1^3$ $\neg s$ $\neg d_2^3$ d_1 d_2
H8

Figure 3.1: Extended conflict hypergraph of I_ψ for $\psi = \forall x_1, x_2, x_3.\exists x_4, x_5.(\neg x_1 \vee x_4 \vee x_2) \wedge (x_3 \vee \neg x_5 \vee \neg x_2)$.

The query used in the reduction is $Q = \neg s$. We claim that

$$\models \psi \iff I_\psi \models_F Q,$$

where $\models \psi$ denotes that ψ is *true*.

$\boxed{\Rightarrow}$ Assume $\models \psi$ and suppose there exists a repair $I' \in \mathit{Rep}(I_\psi, F)$ such that $s \in I'$. We observe the following facts:

(i) $\mathbb{O} \in I'$ (from $<_{I_\psi}$-minimality),

(ii) for $i \in \{1, \ldots, n\}$ either v_i or $\bar{v}_i$ belongs to I' (from $<_{I_\psi}$-minimality),

(iii) for $i \in \{n+1, \ldots, n+m\}$ neither v_i nor $\bar{v}_i$ belongs to I' (from (H2)).

(iv) $\{d_1, \ldots, d_k\} \subseteq I'$ (from $<_{I_\psi}$-minimality).

We construct a valuation V_1 of universal variables (well defined by (ii)):

$$V_1(x_i) = \begin{cases} true & \text{if } v_i \in I', \\ false & \text{if } \bar{v}_i \in I'. \end{cases}$$

Since $\models \psi$ there exists a valuation V_2 of existential variables such that $V_1 \cup V_2 \models \varphi$. Using $V = V_1 \cup V_2$ we construct the following instance:

$$\begin{aligned} I'' = \{\mathbb{O}\} &\cup \{v_i | i \in \{1, \dots, n+m\} \wedge V(x_i) = true\} \\ &\cup \{\bar{v}_i | i \in \{1, \dots, n+m\} \wedge V(x_i) = false\} \\ &\cup \{d_j^p | j \in \{1, \dots, k\} \wedge p \in \{1,2,3\} \wedge V \models l_{j,1} \vee \dots \vee l_{j,p}\} \\ &\cup \{d_1, \dots, d_k\}. \end{aligned}$$

We make two claims about I'':

a) $I'' \models F$,

b) $I'' <_{I_\psi} I'$.

Proof a) The fulfillment of rules (H1), (H2), (H3), and (H4) follows trivially from the construction of I'': I'' contains $\mathbb{O}$ and only one from $\{v_i, \bar{v}_i\}$ for every $i \in \{1, \dots, n+m\}$, and s does not belong to I''. W also note note that $d_{j'}^1 \in I''$ if and only if $V \models l_{j',1}$. This is equivalent to $L_{j',1} \in I''$, and hence (H5) is satisfied. The fulfillment of rules (H6) and (H7) is proved analogously. Finally, to show that the rule (H8) is satisfied note that for every $V \models d_j$ and therefore $d_j^3 \in I''$. At the same time $d_j \in I''$ and $s \notin I''$.

b) First we note that

$$\begin{aligned}\Delta(I'', I_\psi) = &\{v_i | i \in \{1, \ldots, n+m\} \wedge V_1(x_i) \neq \mathit{true}\} \cup \\ &\{\bar{v}_i | i \in \{1, \ldots, n+m\} \wedge V_1(x_i) \neq \mathit{false}\} \cup \\ &\{v_i | i \in \{1, \ldots, n+m\} \wedge V_2(x_i) \neq \mathit{true}\} \cup \\ &\{\bar{v}_i | i \in \{1, \ldots, n+m\} \wedge V_2(x_i) \neq \mathit{false}\} \cup \\ &\{d_j^p | j \in \{1, \ldots, k\} \wedge p \in \{1, 2, 3\} \wedge V \not\models l_{j,1} \vee \ldots \vee l_{j,p}\}.\end{aligned}$$

Take any tuple t from $\Delta(I'', I_\psi)$ and consider the following cases:

- If t is a v_i-tuple for some $i \leq n$, then $V_1(x_i) = \mathit{false}$. From construction of V_1 this implies that $\bar{v}_i \in I'$ and hence $v_i \notin I'$. Therefore $v_i \in \Delta(I', I_\psi)$.

  We handle the case where t is a $\bar{v}_i$-tuple for some $i \leq n$ analogously.

- If t is a v_i- or $\bar{v}_i$-tuple for some $i > n$, then we recall that I' contains neither v_i nor $\bar{v}_i$ (by (iii)) and hence $t \in \Delta(I', I_\psi)$.

- If t is a d_j^1-tuple for some j, then by (H5) $L_{j,1} \notin I''$ and $V \not\models l_{j,1}$. If $l_{j,1}$ is a literal using an existential variable x_i ($i > n$), then neither v_i-tuple nor $\bar{v}_i$ belong to I'. Consequently, $L_{j,1} \notin I'$ and $d_j^1 \notin I'$. If $l_{j,1}$ is a literal using a universal variable x_i ($i \leq n$), then from the construction of V_1 and V we have that $L_{j,1} \notin I'$ and by (H5) $d_j^1 \notin I'$.

  We handle the cases where t is a d_j^2- and d_j^3-tuple analogously.

Finally, we observe that $s \in \Delta(I', I_\psi)$ and $s \notin \Delta(I'', I_\psi)$ which shows that $\Delta(I'', I_\psi) \subset \Delta(I', I_\psi)$.

We finish by observing that a) and b) contradicts I' being a repair of I_ψ.

$\boxed{\Leftarrow}$ Assume $I_\psi \models_F Q$ and suppose $\not\models \psi$, i.e. for some valuation V_1 of universal variables there is no valuation of existential variables which together with V_1 satisfies φ.

We claim that the following instance I' is a repair of I_ψ:

$$\begin{aligned} I' = \{\mathbb{O}, s\} &\cup \{v_i | i \in \{1, \ldots, n\} \wedge V_1(x_i) = true\} \\ &\cup \{\bar{v}_i | i \in \{1, \ldots, n\} \wedge V_1(x_i) = false\} \\ &\cup \{d_j^p | V_1 \models l_{j,q} \text{ for some } q < p \text{ s.t. } l_{j,q} \text{ uses a universal variable}\} \\ &\cup \{d_1, \ldots, d_k\}. \end{aligned}$$

We show that $I' \models F$. The rules (H1), (H2), (H3), (H4), and (H8) are satisfied trivially from the construction of I'. To prove the rule (H5) take any $j \in \{1, \ldots, k\}$ such that $l_{j,1} \in I'$. From the construction of I' this implies that $l_{j,1}$ uses a universal variable and $V \models l_{j,p}$. Hence, $d_{j,1} \in I'$. The fulfillment of the rules (H6) and (H7) is shown analogously.

Next, we show that I' is $<_{I_\psi}$-minimal by contradiction. Suppose there exists an instance I'' such that $I'' \models F$ and $I'' <_{I_\psi} I'$. We take I'' to be $<_{I_\psi}$-minimal. Because for each $i \in \{1, \ldots, n\}$ the instance I' contains either v_i or $\bar{v}_i$ and the instance I'' fulfills (H1) the instance I' and I'' agree on the tuples $v_1, \bar{v}_1, \ldots, v_n, \bar{v}_n$. Also $\{d_1, \ldots, d_k\} \subseteq I''$. We consider the following two cases:

- $s \in I''$. By (H2) the instance I'' does not contain any of the existential variables, i.e. the instances I' and I'' agree on all variables. Therefore $I'' <_{I_\psi} I'$ only if for some j the tuple d_j^p belongs to I'' but not to I'. If $p = 1$, then by (H5) the instance I'' contains $L_{j,1}$. W.l.o.g. we assume that $l_{j,1} = x_q$ for some q, i.e. I'' contains v_q. Because I'' does not contains any of the existential variables, $q \leq n$. From the construction of I' we observe that $d_{j,1} \notin I'$ implies that $V_1 \not\models x_q$. This in turn implies that $\bar{v}_q \in I'$ and because I' and I'' agree on the universal variables, $\bar{v}_q$. This means that I'' violates (H1); a contradiction. We prove the claim for $p > 1$ analogously.

- $s \notin I''$. From $<_{I_\psi}$-minimality for every $i \in \{1, \ldots, n + m\}$ either v_i or $\bar{v}_i$ belongs to I'': if for some i neither v_i nor $\bar{v}_i$ belongs to I'', then $I'' \cup \{v_i\}$ is also consistent and

$I'' \cup \{v_i\} <_{I_\psi} I''$. Consider the following valuation of all variables:

$$V(x_i) = \begin{cases} true & \text{if } v_i \in I'', \\ false & \text{if } \bar{v}_i \in I''. \end{cases}$$

We note that V extends the valuation V_1 and we claim that $V \models \varphi$. Suppose otherwise, i.e. that there exists a conjunct c_j which is not satisfied by V. Assume that d_j^1 belongs to I''. Then by (H5) so does $L_{j,1}$, which implies that $V \models l_{j,1}$; a contradiction; Similarly we show that neither of the tuples d_j^2 and d_j^3 belongs to I''. Since $s \notin I''$ then $d_j \notin I''$ or (H8) is violated. This however contradicts with our earlier observation that $\{d_1, \ldots, d_k\} \subseteq I''$.

We finish the proof of Π_p^2-hardness with the observation that the presented reduction can be implemented in time polynomial in the size of the formula ψ. □

Corollary 3.6 *Repair checking for universal constraints is coNP-complete. The reduction uses a set of three constraints: one functional dependency, one denial constraint with two atoms, and one universal constraint with one atom in its body and 2 atoms in its head.*

Proof To show that the problem is in coNP we note that for given instances I and I' we check if I' is a repair of I by:

1. checking if I' is consistent (can be easily preformed in time polynomial in the size of I');
2. nondeterministically checking if for every independent set M of $G(I, F)$ we have that $M^+ \not<_I I'$.

Soundness and completeness of this procedure follows from Lemma 3.4.

We prove coNP-hardness by reducing $\mathcal{B}_F$ to the complement of 3SAT. We use the transformation from Theorem 3.5 by treating a 3CNF formula as a $\forall^*\exists^* QBF$ formula with an empty sequence of universally quantified variables. Let I_φ be the instance obtained from a 3CNF formula φ. We claim that

$$\varphi \notin 3SAT \iff I'_\varphi \in Rep(I_\varphi, F),$$

where $I'_\varphi = \{d_1, \ldots, d_k, s\}$. The proof of this claim is analogous to the proof of Theorem 3.5.
□

3.3 Complexity of full tuple-generating dependencies

In this section we focus on consistent query answers in the presence of full tuple-generating dependencies and denial constraints.

Theorem 3.7 *The repair checking for full tuple-generating dependencies is in PTIME.*

To prove this theorem we first show an alternative characterization of a repair.

Lemma 3.8 *For a set F of full TGDs and denial constraints and an instance I, a set of tuples I' is a repair of I w.r.t. F if and only if the following conditions are satisfied:*

(i) I' is consistent,

(ii) $T^*_F(I' \cap I) = I'$,

(iii) for every $t \in I \setminus I'$ if $T^*_F(I' \cup \{t\})$ is consistent, then $T^*_F(I' \cup \{t\}) \setminus I \not\subseteq I' \setminus I$.

Proof For the *only if* part take any $I' \in Rep(I, F)$ and note that *(i)* holds trivially.

To show *(ii)* we note that $T^*_F(I \cap I') \subseteq I'$ because I' is consistent (and in particular it satisfies all full TGDs). We also note that $T^*_F(I \cap I')$ is consistent; as the result of T^*_F it satisfies all full TGDs from F and it does not violate a denial constraint or $I' \supseteq T^*_F(I \cap I')$ violates the constraint as well. Therefore, $I' \subseteq T^*_F(I \cap I')$ or I' is not $<_I$-minimal.

To show *(iii)* suppose that there exists a tuple $t \in I \setminus I'$ such that $T^*_F(I' \cup \{t\})$ is consistent and $T^*_F(I' \cup \{t\}) \setminus I \subseteq I' \setminus I$. Then $J_2 <_I I'$ which implies I' is not a repair; a contradiction.

For the *if* part suppose there exists a repair $I'' \in Rep(I, F)$ such that $I'' <_I I'$. This implies that $I' \cap I \subset I'' \cap I$, and consequently $I' \cap I \subseteq I''$. Because I'' is consistent, also $T^*_F(I' \cap I) \subseteq I''$ which by *(ii)* gives us $I' \subseteq I''$. Using again $I'' <_I I'$ we get $I'' \setminus I' \subseteq I$. $I'' \setminus I'$ is non-empty or I'' and i' coincide. Take then any tuple $t \in I'' \setminus I'$ and note that $t \in I \setminus I'$. We also note that $I' \cup \{t\} \subseteq I''$, hence $T^*_F(I' \cup \{t\})$ is consistent, and $T_F(I' \cup \{t\}) \subseteq I''$. By

(iii) we have that $I'' \setminus I \not\subseteq I' \setminus I$. This, however, contradicts $I'' <_I I'$. □
It is easy to see that the characterization of repair in Lemma 3.8 can be used to perform repair checking in polynomial time. This proves Theorem 3.7.

Theorem 3.9 *The consistent query answering in the presence of full tuple-generating dependencies is coNP-complete. The reduction uses a ground atomic query and a cyclic set of three constraints: one FD and two full tuple-generating dependencies.*

Proof The membership of $D_{F,Q}$ to coNP follows from the definition of consistent query answer and Theorem 3.7.

We show coNP-hardness by reducing the complement of $3COL$ to $D_{F,Q}$. $3COL$ is a classic NP-complete decision problem of testing weather a graph has a 3-coloring [Pap94]. 3-coloring is an assignment of one of 3 colors to each of the vertices so that no two adjacent vertices have the same color. Take then any graph $G = (V, E)$ and let $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. For technical reasons we assume that G has no isolated vertices (i.e. vertices incident to no edge).

We construct an instance over the schema consisting of 3 relation names: $R(V, C, E, E')$, $P(E, E')$, and $Q(E)$. The set F of integrity constraints consists of one FD:

$$R : V \to C$$

and two full tuple-generating dependencies

$$R(V_1, C_1, E_1, E_1') \land R(V_2, C_2, E_2, E_2') \land E_1 = E_2 \land C_1 \neq C_2 \to P(E_1, E_1'),$$
$$P(E_1, E_1') \land Q(E_2) \land E_1 = E_2 \to Q(E_1').$$

We use the following types of tuples in the reduction:

- $x_{i,j}^k = R(i, k, j, j+1)$ which corresponds to a copy of the vertex v_i with color k and incident to the edge e_j (we consider a separate copy for each edge incident to the vertex);

- $y_j = P(j, j+1)$ which corresponds to the edge e_j connecting properly colored vertices (vertices of two different colors);

- $q_j = Q(j)$ which corresponds to edges $e_1, \ldots, e_{j-1}$ connecting properly colored vertices.

The set of integrity constraints can be instantiated into the following set of implications:

- for any $i \in \{1, \ldots, n\}$, any $j_1, j_2 \in \{j | j \in \{1, \ldots, m\} \wedge x_i \in e_j\}$, and any two *different* colors k_1 and k_2:
$$x_{i,j_1}^{k_1} \wedge x_{i,j_2}^{k_2} \rightarrow false \tag{C1}$$

- for any $i_1, i_2 \in \{1, \ldots, n\}$, j such that $e_j = \{v_{i_1}, v_{i_2}\}$, and any color $k \in \{1, 2, 3, \}$
$$x_{i_1,j}^{k} \wedge x_{i_2,j}^{k} \rightarrow y_j \tag{C2}$$

- for any $j \in \{1, \ldots, m\}$
$$y_j \wedge q_j \rightarrow q_{j+1} \tag{C3}$$

For G we construct the following instance:

$$I_G = \{x_{i,j}^k | \text{for } i \in \{1, \ldots, n\},\ j \in \{1, \ldots, m\}, \text{ and } k \in \{1, 2, 3\} \text{ such that } v_i \in e_j\} \cup \{q_1\}.$$

The query used in this reduction is $Q = \neg q_{m+1}$. We claim that:

$$I_G \models_F Q \iff G \notin 3\text{COL}.$$

$\boxed{\Rightarrow}$ Suppose $G \in 3\text{COL}$, take any 3-coloring $f : V \rightarrow \{1, 2, 3\}$ of G, and consider the following instance:

$$I' = \{x_{i,j}^{f(v_i)} | \text{ for } i \in \{1, \ldots, n\} \text{ and } j \in \{1, \ldots, m\} \text{ such that } v_i \in e_j\} \cup$$
$$\{y_1, \ldots, y_m, q_1, \ldots, q_{m+1}\}.$$

We claim that I' is a repair. Naturally, I' is consistent. Suppose that I' is not $<_{I_G}$-minimal consistent instance, i.e. there exists an instance I'' such that $I'' <_{I_G} I'$. Obviously, I' and I'' must agree on tuples from I_G. Hence I'' contains tuples q_1 and $x_{i,j}^{f(v_i)}$ for all i, j, and $v_i \in e_j$. I'' cannot contain $x^k_{i,j}$ unless $k = f(v_i)$ or I'' violates (C1). Because f is a 3-coloring, I'' also contains $y_1, \ldots, y_m$ or I'' is violates (C2). Because I'' contains the tuples q_1 and $y_1, \ldots, y_m$ the instance I'' also contains $q_2, \ldots, q_{m+1}$ or I'' is not consistent with rule (C3). This shows that $I'' = I'$; a contradiction. Finally, w note that $I' \not\models Q$ which implies that $I_G \not\models_F Q$.

$\boxed{\Leftarrow}$ Suppose there exists a repair I' such that $q_{m+1} \in I'$. $<_I$-minimality implies that I' has $x^k_{i,j}$ tuples for each $i \in \{1, \ldots, n\}$ and consistency of I' implies that for every i all $x^k_{i,j}$ tuples have the same color (k). Therefore the following function is properly defined:

$$f(v_i) = k \text{ such that } x^k_{i,j} \in I' \text{ for some } j.$$

By (C3) $q_{m+1} \in I'$ and $<_{I_G}$-minimality of I' we have that that $\{q_1, \ldots, q_{m+1}\} \subseteq I'$ and $\{y_1, \ldots, y_m\} \subseteq I'$. By (C1) and (C2) this means that no edge connects vertices with the same color assigned by f. Hence, f is a 3-coloring of G and $G \in$ 3COL; a contradiction.

We finish the proof with the observation that the presented reduction can be implemented in time polynomial in the size of the input graph G. □

3.4 Complexity of acyclic full tuple-generating dependencies

In this section we show that consistent query answering is tractable if we consider acyclic sets of integrity constraints. We assume a fixed acyclic set of full TGDs and denial constraints F.

Recall that a tuple is *base* if it belongs to t; otherwise the tuple is *non-base*. Also, if $t_1 \wedge \ldots \wedge t_n \rightarrow s$, then we call the set of tuples $\{t_1, \ldots, t_n\}$ a *premise* of s.

Definition 3.10 (Support) A *support* of a tuple t is a set of base tuples defined as follows:

1. if $t \in I$, then $\{t\}$ is the only support of t;

2. if $t \notin I$ and $t_1 \wedge \ldots \wedge t_n \rightarrow t$, then for every support $S_1, \ldots, S_n$ of $t_1, \ldots, t_n$ resp. the set $S_1 \cup \ldots \cup S_n$ is a support of t.

$\mathcal{S}(t)$ is the set of all supports of t.

Definition 3.11 (Depth of a support) Given a tuple t, the *depth* of a support S of t is the minimum of the following values:

1. 0 if t is a base tuple, or
2. $k+1$ if $t_1 \wedge \ldots \wedge t_n \rightarrow t$, the sets $S_1, \ldots, S_n$ are supports of $t_1, \ldots, t_n$ respectively, $S = S_1 \cup \ldots \cup S_n$, and k is the maximum depth of the supports $S_1, \ldots, S_n$.

Example 3.12 Consider the schema $\mathcal{S} = \{R(A,B), P(B,C), Q(A,C)\}$ with the set of integrity constraints $F = \{R(x,y) \wedge P(y,z) \rightarrow Q(x,z)\}$ and the instance

$$I = \{R(1,2), R(1,3), P(2,1), P(3,1)\}.$$

Naturally, for any tuple $t \in I$ the only support of t is the set $\{t\}$. However, for the tuple $P(1,1)$ (which belongs to $H(I,F)$) the supports are: $\{R(1,2), P(2,1)\}$ and $\{R(1,3), R(3,1)\}$.

Lemma 3.13 *For every repair I' and every tuple t*

$$t \in I' \iff \exists S \in \mathcal{S}(t). S \subseteq I'.$$

Proof We fix the repair I'.

$\boxed{\Leftarrow}$ We prove the implication by induction on the depth of support.

1. The implication is trivial for any support of depth 0.
2. Assume that the hypothesis holds for every support of depth $< k$ and take a support S of t such that $S \in \mathcal{S}(t)$ and the depth of S is k. We know that there exist tuples $t_1, \ldots, t_n$ with their resp. supports $S_1, \ldots, S_n$, each of depth $< k$, such that $t_1 \wedge \ldots \wedge t_n \rightarrow t$ and $S = S_1 \cup \ldots \cup S_n$. Therefore $S_i \subseteq I'$ and consequently $t_i \in I'$. Thus $t \in I'$ or I' violates $t_1 \wedge \ldots \wedge t_n \rightarrow t$.

$\boxed{\Rightarrow}$ We construct the following set:

$$T = \{t \in I' | \forall S \in \mathcal{S}(t). S \not\subseteq I'\}.$$

We make the following claims:

1. T is a set of non-base tuples. Suppose otherwise and take any base tuple $t \in T$. The only support of this tuple is $\{t\}$ which obviously is included in I' and hence $t \notin T$; a contradiction.

2. T contains no tuple t whose premise is contained in $I' \setminus T$. Otherwise, if $t_1 \wedge \ldots \wedge t_n \rightarrow t$ and $t_i \in I' \setminus T$ for every i, then there exist supports $S_1, \ldots, S_n$ of $t_1, \ldots, t_n$ resp. and all of them are contained in I'. Thus the support $S_1 \cup \ldots \cup S_n$ of t is also contained in I' and hence $t \notin T$; a contradiction.

3. T is empty. Otherwise, consider the instance $I'' = I' \setminus T$. Because T contains only non-base tuples, we have that $I'' <_I I'$. Also, because T contains only tuples with no premise in I' the instance I'' is consistent. Suppose otherwise, i.e. there exists a conflict e in I''. e cannot be a denial conflict or else e would also be present in I'. Thus $e = \{t_1, \ldots, t_n, \neg s\}$, $t_1, \ldots, t_n$ belong to I'' and consequently to I', and s belongs to T. Then, however, s has a premise in I'; a contradiction.

□

Definition 3.14 (Block) A *block* of a tuple t is a pair consisting of a set of base tuples and a set of non-base tuples defined as follows:

1. if $t \notin I$, then $(\varnothing, \{t\})$ is the only block of t;

2. if $t \in I$ and there is a conflict $\{t, t_1, \ldots, t_n\}$, then for every support $S_1, \ldots, S_n$ of $t_1, \ldots, t_n$ resp. the set $(S_1 \cup \ldots \cup S_n, \varnothing)$ is a block of t;

3. if $t \in I$ and $t \wedge t_1 \wedge \ldots \wedge t_n \rightarrow s$, then for every support of $S_1, \ldots, S_n$ of $t_1, \ldots, t_n$ resp. and every block (B, N) of s the pair $(S_1 \cup \ldots \cup S_n \cup B, N)$ is a block of t.

$\mathcal{B}(t)$ is the set of all blocks of t.

Definition 3.15 (Depth of a block) Given a tuple t, the *depth* of a block (B, N) of t is the minimum of the following values:

1. 0 if $N = \varnothing$,

2. $k+1$ if $t \wedge t_1 \wedge \ldots \wedge t_n \rightarrow s$, $S_1, \ldots, S_n$ are supports of $t_1, \ldots, t_n$ respectively, (B', N) is a block of s of depth k, and $B = S_1 \cup \ldots \cup S_n \cup B'$.

Example 3.16 Consider the schema $\mathcal{S} = \{R(A,B), P(A,B)\}$ with the set of integrity constraints $F = \{R(x,y) \rightarrow P(x,y), P : A \rightarrow B\}$ and the instance $I = \{R(1,1), P(1,1), P(1,2)\}$. The only block of $P(1,1)$ is $(\{P(1,2)\}, \varnothing)$ and the only block of $P(1,2)$ is $(\{P(1,1)\}, \varnothing)$. The only block of $R(1,1)$ is $(\{P(1,2)\}, \varnothing)$.

Lemma 3.17 *If F is acyclic, then for every repair I' and every tuple t*

$$t \notin I' \iff \exists (B,N) \in \mathcal{B}(t). B \subseteq I' \wedge N \cap I' = \varnothing.$$

ProofWe fix a repair I'.

$\boxed{\Leftarrow}$ We prove the implication by induction on the depth of a block:

1. Trivial for blocks of depth 0.

2. Suppose the hypothesis holds for every block of depth $< k$ and take any block (B,N) of t of depth k such that $B \subseteq I'$ and $N \cap I' = \varnothing$. From the construction of (B,N) there exist tuples $t_1, \ldots, t_n, s$ such that $t \wedge t_1 \wedge \ldots \wedge t_n \rightarrow s$. Also, there exists a block (B_s, N_s) of s of depth $< k$ such that $B = S_1 \cup \ldots \cup S_n \cup B_s$ and $N = N_s$. Because $B \subseteq I'$, the tuples $t_1, \ldots, t_n$ are present in I', and because additionally $N \cap I' = \varnothing$, the tuple s is not present in I'. Therefore t is not present in I' or I' is inconsistent.

$\boxed{\Rightarrow}$ For a tuple $t \notin I$, a block (B,N) is *proper* if $B \subseteq I'$ and $N \cap I' = \varnothing$. Let's select the set of tuples that don't have a proper block:

$$T = \{t \notin I' | \forall (B,N) \in \mathcal{B}(t). B \not\subseteq I' \vee N \cap I' \neq \varnothing\}$$

We claim the following:

1. $T \subseteq I$, i.e. T contains only base tuples. Suppose otherwise that there exists a tuple $t \in T \setminus I$. Then consider the only block $(\varnothing, \{t\})$ of t. Obviously, it is a proper block; a contradiction.

2. No tuple from T creates a denial conflict with tuples from I'. Otherwise, if some $t \in T$ and some $t_1, \ldots, t_n \in I'$ create a denial conflict $\{t, t_1, \ldots, t_n\}$, then for any supports $S_1, \ldots, S_n$ of $t_1, \ldots, t_n$ resp. the pair $(S_1 \cup \ldots \cup S_n, \varnothing)$ is a proper block of t and consequently $t \notin T$; a contradiction.

3. For every $t \in T$ there exist $t_1, \ldots, t_n \in I'$ such that $t \wedge t_1 \wedge \ldots \wedge t_n \to s$ for some $s \in T$. To prove it we make the following observations:

   (a) Suppose there exists $t \in T$ for which there is no such $t_1, \ldots, t_n \in I'$ that $t \wedge t_1 \wedge \ldots \wedge t_n \to s$ for any s. Then by 2 above the instance $I' \cup \{t\}$ is consistent and by 1 above $I' \cup \{t\} <_I I'$; a contradiction.

   (b) Suppose there exists $t \in T$ such that for every $t_1, \ldots, t_n \in I'$ if $t \wedge t_1 \ldots \wedge t_n \to s$, then $s \in I'$. Then again $I' \cup \{t\}$ is a consistent instance and $I' \cup \{t\} <_I I'$; a contradiction.

   (c) Suppose there exists $t \in T$ such that there exists $t_1, \ldots, t_n \in I'$ such that $t \wedge t_1 \wedge \ldots \wedge t_n \to s$ and $s \notin T$. By (b) s also does not belong to I'. Therefore there exists a block (B, N) of s such that $B \subseteq I'$ and $N \cap I' = \varnothing$. Then the block of t constructed in case 3 of Definition 3.14 using $s, t_1, \ldots, t_n$ is proper and hence $t \notin T$; a contradiction.

4. $T = \varnothing$. Otherwise we construct an infinite sequence $s_0, s_1, \ldots,$ of tuples from T as follows. For s_0 select any tuple of T. For $i > 0$ as s_i choose any tuple s from T such that $s_{i-1} \wedge t_1 \wedge \ldots \wedge t_n \to s$ for any $t_1, \ldots, t_n$ from I'. Such tuple exists by 3 above. Now, let $R_{j_0}, R_{j_1}, \ldots$ be the sequence of relation names of tuples $s_0, s_1, \ldots$ respectively. Naturally, for every i there is an edge in the dependency graph $\mathcal{D}(F)$ coming from R_{j_i} to $R_{j_{i+1}}$. Since $\mathcal{D}(F)$ has a finite number of vertices, it must have a cycle. This implies that F is not acyclic; a contradiction.

□

Lemma 3.18 *If F is acyclic, then for any tuple t the sizes of $\mathcal{S}(t)$ and $\mathcal{B}(t)$ are polynomial in the size of I.*

Proof First we observe that because the set of constraints is acyclic, the depth of all supports and all blocks is bound by a constant (equal to the number of different relation names). With a simple induction by the depth of a support we can show that the size of any support of depth k is $O(n^k)$, where n is the maximum number of a atoms in a head of a full TGD in F. Because we assume F to be fixed, the size of any support is bound by a constant $K = n^k$. The number of different subsets of tuples from I is therefore $O(I^K)$. We prove the claim for blocks analogously. $\square$

Lemma 3.19 *For any set of base tuples P and any set of non-base tuples N a repair containing all tuples of P and disjoint with N exists if and only if $T_F^*(P)$ is consistent and disjoint with N.*

Proof The *only if* part of the proof is trivial. For the *if* part consider a set $\mathcal{C}$ of consistent instances that contain P and that are disjoint with N. Because $T_F^*(P)$ is consistent and disjoint with N this set is nonempty ($T_F^*(P)$ is one of its elements). Now, from $\mathcal{C}$ we select any $<_I$-minimal instance I'. We claim that I' is a repair. Suppose otherwise, i.e. there exists a consistent instance I'' such that $I'' <_I I'$. We observe that:

1. $P \subseteq I''$ or otherwise $I'' \not<_I I'$ (because P is a subset of base tuples).
2. $T_F^*(P) \subseteq I''$ or otherwise I'' is not consistent.
3. $N \cap I'' \neq \varnothing$ or otherwise I' is not a $<_I$-minimal element of $\mathcal{C}$.

Now, take any $t \in N \cap I''$ and recall that $N \cap I = \varnothing$. Thus $t \in \Delta(I, I'')$ but $t \notin \Delta(I, I')$. This contradicts $I'' <_I I'$. We finish the proof with the observation that I' contains $T_F^*(P)$ and is disjoint with N. $\square$

Theorem 3.20 *Consistent query answering is in PTIME for any acyclic set of full tuple-generating dependencies and denial constraints and any ground quantifier-free query.*

Proof We extend the algorithm computing consistent query answer in the presence of denial constraints from [CM05] to handle acyclic sets of constraints.

Take any query Q and assume it is in CNF, i.e.

$$Q = Q_1 \land \ldots \land Q_w,$$

where Q_l is a disjunction of literals. We note that *true* is not the consistent query answer to Q if and only if there exists a repair I' such that $I' \models \neg Q_l$ for some $l \in \{1, \ldots, w\}$. The algorithm attempts to find if there exists such a repair iteratively for each $l \in \{1, \ldots, w\}$. Fix l and let

$$\neg Q_l = t_1 \land \ldots \land t_n \land \neg s_1 \land \ldots \land \neg s_m. \tag{3.4}$$

We assume that the tuples $t_1, \ldots, t_n, s_1, \ldots, s_m$ belong to $H(I, F)$. Otherwise if $t_i \notin H(I, F)$ for some $i \in \{1, \ldots, n\}$, there is no repair satisfying Q_l and if $s_j \notin H(I, F)$ for some $j \in \{1, \ldots, m\}$, we can remove $\neg s_j$ from the conjunction.

For every $i \in \{1, \ldots, n\}$ the algorithm nondeterministically chooses a support S_i of t_i. Next, for $j \in \{1, \ldots, m\}$ the algorithm nondeterministically chooses a block (B_j, N_j) of s_j. If there is no block for s_j, then the algorithm fails for this l. Finally, the algorithm constructs two sets:

$$P = \bigcup_{i=1}^{n} S_i \cup \bigcup_{j=1}^{m} B_j, \quad \text{and} \quad N = \bigcup_{j=1}^{m} N_j.$$

A repair satisfying $\neg Q_l$ exists if and only if $T_F^*(P)$ is consistent and disjoint with N. To prove soundness and correctness of this procedure we note that by Lemma 3.19, $T_F^*(P)$ is consistent and disjoint with N if and only if there exists a repair I' containing P and disjoint with N. Because of the nondeterministic construction of P and N by Lemma 3.13 and Lemma 3.17 this is equivalent to I' containing $\{t_1, \ldots, t_n\}$ and disjoint with $\{s_1, \ldots, s_m\}$, i.e. $I' \models \neg Q_l$.

We finish the proof with the observation that the algorithm makes a fixed number of nondeterministic choices and every choice is made from a set whose size is polynomial in the size of I. Hence, the algorithm works in time polynomial in the size of I. □

3.5 Related work

For a complete survey of the topic we refer the reader to [Ber06, BC03, Cho07]. Here, we discuss only work focused on the computational complexity of consistent query answers. Discussion of system-oriented work is in Section 4.7.

Our work was inspired by positive results for denial constraints presented in [CM05]. Repairs of databases inconsistent w.r.t. denial constraints are obtained by deletion of some (conflicts) tuples only. In [CM05] the repairs obtained by deletion of tuples only are also used to define consistent query answers in the presence of inclusion dependencies (IND), i.e. constraints of the form

$$\forall \bar{x}_1 \exists \bar{x}_3 R(\bar{x}_1) \rightarrow P(\bar{x}_2, \bar{x}_3),$$

where $\bar{x}_2 \subseteq \bar{x}_1$. An IND of the form above is often denoted by $R[X] \subseteq P[Y]$, where X and Y are the sets of attributes corresponding to $\bar{x}_2$. Also, the IND $R[X] \subseteq P[Y]$ is a *foreign-key constraint* if Y is the *key* of the relation P.

We note that universal constraints capture only *full* INDs, i.e. INDs with no existentially quantified variables. Considering repairs obtained by deleting tuples only is natural in the scenarios like *data warehousing*, where the data is complete but may be incorrect; In particular we can assume that if a fact is not present in the original database, then it is not true.

Example 3.21 Consider the database schema $\mathcal{S} = \{R(A, B), P(C, D)\}$ with the set of integrity constraints $F = \{R : A \rightarrow B, P[D] \subseteq R[A]\}$ and take the following database

$$I = \{R(1, 1), R(1, 2), P(3, 1), P(5, 6)\}.$$

Then, there are two minimal repairs obtained by deleting tuples only:

$$I_1 = \{R(1, 1), P(3, 1)\}$$

and

$$I_2 = \{R(1, 2), P(3, 1)\}.$$

We note that if we restrict the set of integrity constraints to one key dependency per relation and foreign-key constraints, then the discussed approach has an interesting property. Every repair can be obtained in two steps: In the first step we repair only violations of foreign-key constraints which yields exactly one instance; Next, we repair the violations of key dependencies.

Example 3.22 (cont. of Example 3.21) Repairing I w.r.t $P[D] \subseteq R[A]$ yields the instance $I' = \{R(1,1), R(1,2), P(3,1)\}$. Note that consecutive repairing I' w.r.t $R : A \to B$ does not violate the foreign-key constraint because every repair of I' has a tuple $R(1,x)$ for some x. Hence, I_1 and I_2 are repairs (in the sense of Definition 2.4) of I' w.r.t. $R : A \to B$.

Naturally, this observation allows to use any method for computation of consistent query answers in the presence of key constraints. [CM05] also shows that relaxing the restriction on the set of integrity constraints leads to intractability and consistent query answering to arbitrary sets of INDs and FDs becomes Π_2^p-complete.

Obtaining repairs by deletion of tuples only is not necessarily a natural approach in the scenarios where we cannot assume that information missing in the database is untrue, for instance in the context of integration of sources that may be missing some information. Then, we might want to consider standard repairs obtained by deleting and inserting a minimal set of tuples, i.e. repairs in the sense of Definition 2.4. We observe, however, that while in the case of universal constraints the missing tuples that create conflicts are implicitly defined, the presence of existentially quantified variables in INDs leads to possibly infinite number of repairs.

Example 3.23 (cont. of Example 3.21) The set of standard repairs for I w.r.t F consists of the following instances:

$$I_x = \{R(1,1), R(6,x), P(3,1), P(5,6)\} \quad \text{for any } x$$

and

$$I_x = \{R(1,2), R(6,x), P(3,1), P(5,6)\} \quad \text{for any } x.$$

Cali et al. in [CLR03] show that consistent query answering becomes undecidable for arbitrary sets of INDs and FDs. The problem becomes tractable when the set of integrity constraints is restricted to *non-key-conflicting* INDs; IND $R[X] \subseteq P[Y]$ is non-key-conflicting if Y is not a strict superset of the key of P. Then, the problem of consistent query answering is Π_p^2-complete.

Chapter 4

The system Hippo

In this chapter we present the system Hippo for computing consistent query answers to π-free relational algebra queries in the presence of full tuple generating dependencies and denial constraints.

4.1 System architecture

The flow of data in the system is presented in Figure 4.1. The only output of the system

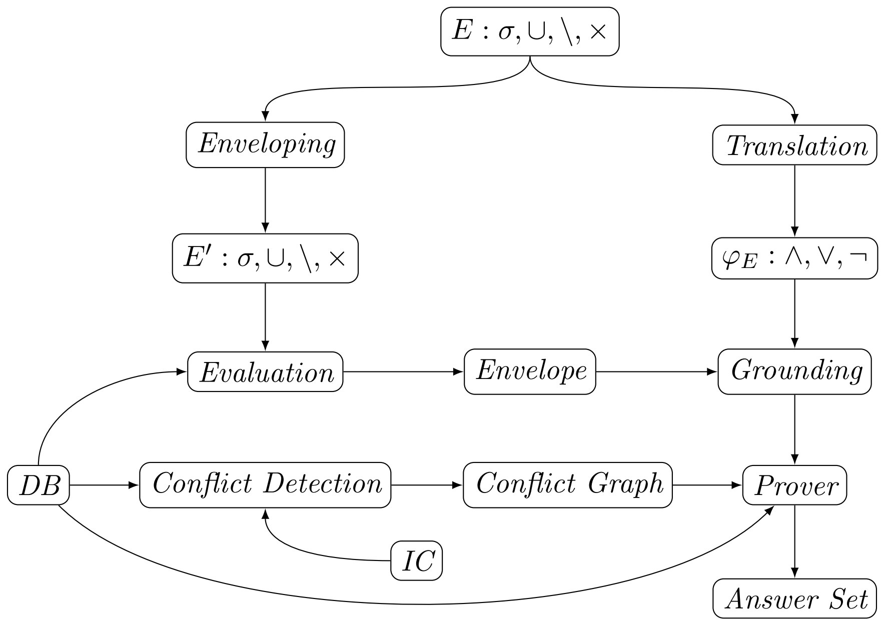

$E : \sigma, \cup, \setminus, \times$
Enveloping
Translation
$E' : \sigma, \cup, \setminus, \times$
$\varphi_E : \wedge, \vee, \neg$
Evaluation
Envelope
Grounding
DB
Conflict Detection
Conflict Graph
Prover
IC
Answer Set

Figure 4.1: Data flow in the Hippo system

is the *Answer Set* consisting of consistent answers to the input query E in the database instance DB w.r.t. the set of integrity constraints IC.

Before processing any input query, the system performs *Conflict Detection* and creates the *Conflict Hypergraph*. In Section 4.2 we observe that for the purposes of computing consistent query answers we do not need to store the whole extended hypergraph, but only hyperedges representing conflicts relevant to computing consistent query answers.

The processing of a query E starts with *Enveloping* it to obtain a query E' which after *Evaluation* returns an *Envelope*, a superset of the consistent query answers to E. Next, the system performs the *Translation* of the query E into a quantifier-free first-order logic formula φ_E. We use this formula to perform *Grounding* (ground substitution) for every tuple from the *Envelope* yielding a ground quantifier-free query. For such a query the *Prover* uses the algorithm from Section 3.4 to check if *true* is the consistent answer to the query. Consequently, depending if the result of this check is positive or not the tuple is placed into the *Answer Set* or discarded.

We note that the *Prover* does not need a direct knowledge of IC; all needed information is stored in the *Conflict Hypergraph*. We recall, however, from Section 3.4 that to check if *true* is the consistent answer to φ_E grounded with a tuple *Prover* needs to find out if other tuples are present in the database.

4.2 Conflict hypergraph construction

We start by identifying tuples that may be inserted into a repair to resolve inconsistencies. From now on we assume a fixed acyclic set F of full TGDs and denial constraints.

4.2.1 Hull, core, and shell

Definition 4.1 (Hull) For any $R \in \mathcal{S}$ the *hull* expression for R in the presence of F, denoted $H(R, F)$, is an expression consisting of the union of R and the union of the following expressions for every rule in F of the form $R_{i_1}(\bar{x}_1) \wedge \ldots \wedge R_{i_m}(\bar{x}_m) \wedge \rho \rightarrow R(\bar{y})$:

$$\pi_Y(\sigma_{\underline{\rho}}(H(R_{i_1}, F) \times \ldots \times H(R_{i_m}, F))),$$

where Y are the positions of $\bar{y}$ variables and $\underline{\rho}$ is the selection expression obtained from ρ and a conjunction of equality (join) conditions for variables that repeat among $\bar{x}_i$ vectors.

Given an instance I, its *hull* w.r.t F, denoted $H(I,F)$, is the union of all hull expressions evaluated over I, i.e. formally:

$$H(I,F) = \{R(t) | R \in \mathcal{S} \wedge t \in QA(I, H(R,F))\}.$$

We note that because F is acyclic, there is no cycle in the definitions of hull expressions. We also note that the hull presented here coincides with the more general version in Definition 3.1. We present the expressions to emphasize that for for an acyclic set of full TGDs and denial constraints the hull can be constructed with SQL expressions (use of recursive Datalog is not necessary). We also note that for acyclic TGDs and denial constraints the hull can be characterized as follows.

Fact 4.2 *For any instance I and set of integrity constraints F, the hull $H(I,F)$ is the minimal set of tuples such that :*

1. *$I \subseteq H(I,F)$ and*
2. *if $t_1 \wedge \ldots \wedge t_n \rightarrow t$ and $\{t_1, \ldots, t_n\} \subseteq H(I,F)$, then $t \in H(I,F)$.*

Example 4.3 Consider a schema consisting of two binary relation names $R(A,B)$ and $P(A,B)$, and the following (acyclic) set of integrity constraints

$$F = \{R : A \rightarrow B, P : A \rightarrow B, R(A,B) \wedge R(B,C) \wedge A \leq C \rightarrow P(A,C)\}.$$

The hull expressions are:

$$\begin{aligned} H(R,F) &= R, \\ H(P,F) &= P \cup \pi_{1,4}(\sigma_{1 \leq 4 \wedge 2=3}(R \times R)). \end{aligned}$$

We note, however, that the hull of I is not necessarily the union of all repairs of I.

Example 4.4 We use the schema and the set of constraints F from Example 4.3 and take the instance $I = \{R(1,1), R(1,2), P(1,1)\}$. The hull is

$$H(I,F) = \{R(1,1), R(1,2), P(1,1), P(1,2)\}.$$

We note that the tuple $P(1,2)$ is not present in either of the repairs of I w.r.t. F: $I_1 = \{R(1,1), P(1,1)\}$ and $I_2 = \{R(1,2), P(1,1)\}$.

Now, we identify the tuples that are not removed from the original instance when constructing a repair.

Definition 4.5 (Core) For any $R \in \mathcal{S}$ the *core* expression for R in the presence of F, denoted $C(R,F)$, is an expression consisting of the difference between R and the union of the following expressions:

- for every denial constraint $R(\bar{x}) \wedge R_{i_1}(\bar{x}_1) \wedge \ldots \wedge R_{i_m}(\bar{x}_m) \wedge \rho \rightarrow \mathit{false}$ in F

  $$\pi_X(\sigma_{\underline{\rho}}(H(R,F) \times H(R_{i_1},F) \times \ldots \times H(R_{i_m},F))),$$

  where X are the positions corresponding to $\bar{x}$ variables and $\underline{\rho}$ is obtained from ρ and a conjunction of equality (join) conditions for repeated variables;

- for every full TGD $R(\bar{x}) \wedge R_{i_1}(\bar{x}_1) \wedge \ldots \wedge R_{i_m}(\bar{x}_m) \wedge \rho \rightarrow P(\bar{y})$ in F

  $$\pi_X(\sigma_{\underline{\rho}}(H(R,F) \times H(R_{i_1},F) \times \ldots \times H(R_{i_m},F) \times (H(P,F) \setminus C(P,F)))),$$

  where X are positions corresponding to $\bar{x}$ variables and $\underline{\rho}$ is obtained from ρ and a conjunction of equality (join) conditions for repeated variables.

Given an instance I, its *core* w.r.t F, denoted $C(I,F)$, is the union of all core expressions evaluated over I, i.e. formally:

$$C(I,F) = \{R(t) | R \in \mathcal{S} \wedge t \in QA(I, C(R,F))\}.$$

Again, we note that because F is acyclic, the definitions of hull expression are not cyclic.

Example 4.6 Take the schema and the set of integrity constraints from of Example 4.3 and recall that the functional dependency $P : A \rightarrow B$ stands for the formula

$$\forall x_1, x_2, x_3, x_4. P(x_1, x_2) \land P(x_3, x_4) \land x_1 = x_2 \land x_2 \neq x_4 \rightarrow false.$$

Similar formula exists for $R : A \rightarrow B$. The core expressions are:

$$\begin{aligned}
C(P, F) &= P \setminus (\pi_{1,2}(\sigma_{1=2\land 3\neq 4}(H(P, F) \times H(P, F)))),\\
C(R, F) &= R \setminus (\pi_{1,2}(\sigma_{1=2\land 3\neq 4}(R \times R))\\
&\qquad \cup \pi_{1,2}(\sigma_{1<4\land 2=3\land 1=5\land 4=6}(H(R, F) \times H(R, F) \times (H(P, F) \setminus C(P, F))))\\
&\qquad \cup \pi_{3,4}(\sigma_{1<4\land 2=3\land 1=5\land 4=6}(H(R, F) \times H(R, F) \times (H(P, F) \setminus C(P, F))))).
\end{aligned}$$

We note that the full TGD produces two subexpressions (but with different projection lists) because the relation name R is present twice in the full TGD. Relation names are present twice also in the formulas expressing the FDs, however, the resulting subexpressions are equivalent and therefore need not to be repeated.

We also observe an alternative characterization of the core.

Fact 4.7 *For any instance I, the core $C(I, F)$ is the maximal set of tuples of I such that:*

1. *if $\{t_1, \ldots, t_n\} \subseteq H(I, F)$ is a denial conflict, then $\{t_1, \ldots, t_n\} \cap C(I, F) = \varnothing$,*
2. *if $t_1 \land \ldots \land t_n \rightarrow s$ and $\{t_1, \ldots, t_n, s\} \subseteq H(I, F)$, then $\{t_1, \ldots, t_n, s\} \cap C(I, F) = \varnothing$.*

We note that, similarly to the hull, the core it is not necessarily the intersection of all repairs, i.e. the core is not the *core instance* commonly used in the literature [ABC+03b, EFGL03].

Example 4.8 We assume the setting of Example 4.4. The core $C(I, F)$ is empty while the tuple $P(1, 1)$ is present in both repairs I_1 and I_2.

Corollary 4.9 *For any database instance I and any repair $I' \in Rep(I, F)$*

$$C(I, F) \subseteq I' \subseteq H(I, F).$$

Proof We get $I' \subseteq H(I,F)$ from Lemma 3.4. Suppose there exists $x \in C(I,F)$ such that $x \notin I'$, then $\{x\} \cup I'$ is consistent (by Fact 4.7) and because $C(I,F) \subseteq I$, $I' \cup \{x\} <_I I'$; a contradiction. □

Definition 4.10 (Shell) Given an instance I its *shell* w.r.t. F, denoted $S(I,F)$, is

$$S(I,F) = H(I,F) \setminus C(I,F).$$

Lemma 4.11 *For any tuple $t \in S(I,F)$ every support and every block of t is contained in $S(I,F)$, i.e.*

$$\forall S \in \mathcal{S}(t). S \subseteq S(I,F),$$
$$\forall (P,N) \in \mathcal{B}(t). P \subseteq S(I,F) \wedge N \subseteq S(I,F).$$

Proof We prove the claim for the supports by induction over the depth of a support (Definition 3.11).

1. Trivial for supports of depth 0.

2. Consider a support S of t of depth $k+1 > 0$. Take the ground rule $t_1 \wedge \ldots \wedge t_n \rightarrow t$ and the supports $S_1, \ldots, S_n$ of $t_1, \ldots, t_n$ respectively such that $S = S_1 \cup \ldots \cup S_n$ and the maximum degree of $S_1, \ldots, S_n$ is k. Then by Facts 4.2 and 4.7 we have $\{t_1, \ldots, t_n\} \subseteq S(I,F)$ and by induction hypothesis $S_1 \cup \ldots \cup S_n \subseteq S(I,F)$.

The claim for blocks is proved by induction over the depth of a block:

1. Trivial for block of depth 0 of the form $(\{t\}, \varnothing)$.

2. For any other block $(B, \varnothing)$ of depth 0, let $\{t, t_1, \ldots, t_n\}$ be a conflict such that $B = S_1 \cup \ldots \cup S_n$, where $S_1, \ldots, S_n$ are the supports of $t_1, \ldots, t_n$. By Fact 4.7 we have $\{t_1, \ldots, t_n\} \subseteq S(I,F)$ and from the claim for supports $S_1 \cup \ldots \cup S_n \subseteq S(I,F)$.

3. For a block (B,N) of depth $k+1 > 0$, let $t \wedge t_1 \wedge \ldots \wedge t_n \rightarrow s$ be such a ground rule that $B = S_1 \cup \ldots \cup S_n \cup B'$, where $S_1, \ldots, S_n$ are supports of $t_1, \ldots, t_n$ respectively and

(B', N) is the block of s of depth k. Then by Facts 4.7 and 4.2 we have $\{t_1, \ldots, t_n, s\} \subseteq S(I, F)$ and by induction hypothesis and the claim for supports we have that $S_1 \cup \ldots \cup S_n \cup B' \subseteq S(I, F)$ and $N \subseteq S(I, F)$.

□

Lemma 4.12 *For any instance I and any $J \subseteq S(I, F)$ we have that $T^*_F(J) \subseteq S(I, F)$.*

Proof By Fact 4.2 $T_F(J) \subseteq H(I, F)$ and by Fact 4.7 $T_F(J) \subseteq S(I, F)$. Hence, $T^*_F(J) \subseteq S(I, F)$. □

4.2.2 Active conflict hypergraph

We use the shell of an instance to identify the conflicts that are necessary to find if *true* is a consistent answer to a ground FOL query.

Definition 4.13 (Active extended conflict hypergraph) The *active* extended conflict hypergraph $G^{\mathcal{A}}(I, F)$ is the subgraph of $G(I, F)$ induced by $S(I, F)$. With every tuple of $S(I, F)$ we also store the information if the tuple is base or not (i.e. if it belongs to I or not).

Now, we revisit the algorithm finding if *true* is the consistent answer to a ground FOL query (Theorem 3.20) and make the following observations:

1. We can assume that the query (3.4)

$$\neg Q_l = t_1 \wedge \ldots \wedge t_n \wedge \neg s_1 \wedge \ldots \wedge \neg s_m$$

   contains only tuples from $S(I, F)$. If for some $i \in \{1, \ldots, n\}$ the conjunct t_i of Q_l belongs to $C(I, F)$ (i.e. it belongs to I and does not belong to $S(I, F)$), then it can be removed from consideration because every repair of I contains t. Similarly, if for some $j \in \{1, \ldots, m\}$ the conjunct $\neg s_j$ of $\neg Q_l$ belongs to $C(I, F)$, then there is no repair satisfying Q_l.

2. The set of supports and the set of blocks of any tuple from $S(I,F)$ can be constructed from $G^{\mathcal{A}}(I,F)$ (Lemma 4.11).

3. The transitive closure of a subset of $S(I,F)$ can be computed using $G^{\mathcal{A}}(I,F)$ only (Lemma 4.12).

4.3 Basic enveloping

We begin by observing that consistent answers to a query need not to be contained in the set of answers to the query.

Example 4.14 Suppose the schema consists of two relation names $R(A,B)$ and $S(A,B,C,D)$. The set of integrity consists of one functional dependency $F = \{R : A \to B\}$. Consider the instance $I = \{R(1,1), R(1,2), S(1,1,1,2)$ and a relational algebra query $E = S \setminus \sigma_{2 \neq 4}(R \times R)$. The set of answers to E is empty, while the set of consistent query answers is $\{(1,1,1,2)\}$.

Therefore to get a superset of consistent query answer we introduce the following expression.

Definition 4.15 (Envelope expression) Given an RA expression E and a set of integrity constraints we define the *envelope* expression of E w.r.t. F, denoted by $Env(E,F)$, as follows (by mutual recursion with an auxiliary expression $Cert(E,F)$):

$$\begin{aligned}
Env(R,F) &= H(R,F), & Cert(R) &= C(R,F),\\
Env(\sigma_\chi(E),F) &= \sigma_\chi(Env(E,F)), & Cert(\sigma_\chi(E),F) &= \sigma_\chi(Cert(E,F)),\\
Env(\pi_X(E),F) &= \pi_X(Env(E,F)), & Cert(\pi_X(E),F) &= \pi_X(Cert(E,F)),\\
Env(E_1 \times E_2,F) &= Env(E_1,F) \times Env(E_2,F), & Cert(E_1 \times E_2,F) &= Cert(E_1,F) \times Cert(E_2,F),\\
Env(E_1 \cup E_2,F) &= Env(E_1,F) \cup Env(E_2,F), & Cert(E_1 \cup E_2,F) &= Cert(E_1,F) \cup Cert(E_2,F),\\
Env(E_1 \setminus E_2,F) &= Env(E_1,F) \setminus Cert(E_2,F), & Cert(E_1 \setminus E_2,F) &= Cert(E_1,F) \setminus Env(E_2,F).
\end{aligned}$$

Proposition 4.16 *For any instance I, any repair $I' \in Rep(I,F)$, and any expression E*

$$QA(I, Cert(E,F)) \subseteq QA(I', E) \subseteq QA(I, Env(E,F)).$$

Proof[by induction over the structure of E] The base case follows from Corollary 4.9. For the monotonic operators σ, π, $\times$, and $\cup$ the thesis holds trivially by induction hypothesis. For the expression involving the set difference operator $E_1 \setminus E_2$ assume by induction hypothesis that

$$QA(I', E_1) \subseteq QA(I, Env(E_1, F)) \quad \text{and} \quad QA(I, Cert(E_2, F)) \subseteq QA(I', E_2).$$

for any $I' \in Rep(I, F)$. Fix the repair I' and note that if $A \subseteq B$ and $C \subseteq D$, then $A \setminus D \subseteq B \setminus C$. Therefore:

$$QA(I', E_1 \setminus E_2) = QA(I', E_1) \setminus QA(I', E_2) \subseteq QA(I, Env(E, F)) \setminus QA(I, Cert(E, F)).$$

Similarly we prove that

$$QA(I, Cert(E_1 \setminus E_2, F)) \subseteq QA(I', E_1 \setminus E_2).$$

□

Recall that $CQA(I, E) = \bigcap_{I' \in Rep(I,F)} QA(I', E)$. Therefore,

Corollary 4.17 *For any instance I and any expression E*

$$QA(I, Cert(E, F)) \subseteq CQA(I, E) \subseteq QA(I, Env(E, F)).$$

4.4 Tuple checks and knowledge gathering

We recall that for every instance of φ_E grounded with a tuple from *Envelope* the *Prover* needs to perform *tuples checks*, i.e. find if other tuples are present in the database instance. A naive implementation performs a membership query for each tuple check. Because in virtually all relational database systems performing a query requires an allocation of a significant amount of resources, this approach may yield a high overhead. To address this issue we show how to avoid some of the tuple check by performing *knowledge gathering*. We show how to extend the envelope expression to provide all information that can be needed by *Prover*.

We start by identifying the tuple checks that can be performed by *Prover*. We obtain the set of facts *relevant* to computing consistent answers to a ground query $\varphi_E(t)$ by following the standard translation of E into φ_E and the grounding of φ_E with a tuple t from *Envelope*.

Definition 4.18 (Relevant facts) For a given π-free expression E and a tuple t compatible with E, the set $TC(E, t)$ of *relevant facts* is defined recursively:

$$\begin{aligned}
TC(R, t) &= \{R(t)\}, \\
TC(\sigma_\chi(E), t) &= TC(E, t), \\
TC(E_1 \times E_2, t_1 \cdot t_2) &= TC(E_1, t_1) \cup TC(E_2, t_2), \\
TC(E_1 \cup E_2, t) &= TC(E_1, t) \cup TC(E_2, t), \\
TC(E_1 \setminus E_2, t) &= TC(E_1, t) \cup TC(E_2, t).
\end{aligned}$$

The following example illustrates that the tuple t used to ground φ_E may allow to infer if some of the relevant facts of $TC(E, t)$ are in the database instance I. Consequently, this information can be used to answer some of the tuple checks without performing membership queries.

Example 4.19 Consider the relational schema consisting of two binary relation names $\mathcal{S} = \{R(A, B), P(A, B)\}$ with the set of integrity constraints $F = \{R : A \to B, P : A \to B\}$ and the following query $E = \sigma_{1=a}(R \times (R \cup P))$. We also have an instance I, but we do not assume any knowledge of the instance. The set of relevant facts is $TC(E, t) = \{R(a, b), R(c, d), P(c, d)\}$.

Because $H(R, F) = R$ and $H(P, F) = P$, the envelope expression coincides with original query, i.e. $Env(E, F) = E$. Now, suppose that a tuple $t = (a, b, c, d)$ is an answer to $Env(E, F)$ in I.

Naturally, $t \in QA(I, E)$ implies that $R(a, b) \in I$. At the same time, although we know that $R(c, d)$ or $P(c, d)$ must be in I, we don't have sufficient information to say exactly if I contains $R(c, d)$, $P(c, d)$, or both.

We attempt to infer information about the state of the database instance based on the results of evaluating the envelope expression and the active conflict hypergraph. We call this process *knowledge gathering.*

Definition 4.20 (Knowledge gathering) For any database instance I, any π-free expression E, and any tuple t compatible with E, the set $KG(E,t)$ is defined as follows:

$$\begin{aligned}
KG(R,t) &= \begin{cases} \varnothing, & \text{if } t \text{ is a non-base tuple of } G^{\mathcal{A}}(I,F),\\ \{R(t)\}, & \text{otherwise,}\end{cases}\\
KG(\sigma_\rho(E),t) &= KG(E,t),\\
KG(E_1 \times E_2, t_1 \cdot t_1) &= KG(E_1,t_1) \cup KG(E_2,t_2),\\
KG(E_1 \cup E_2, t) &= KG(E_1,t) \cap KG(E_2,t),\\
KG(E_1 \setminus E_2, t) &= KG(E_1,t).
\end{aligned}$$

We note that $KG(E,t)$ is always nonempty and its size is linear in the size of the query E. With a simple induction over the structure of the expression we can show

Fact 4.21 *For any instance I, any π-free expression E, and any tuple t compatible with E*

$$KG(E,T) \subseteq TC(E,t).$$

Now, we state that knowledge gathering is a sound derivation of facts holding in the database.

Theorem 4.22 (Soundness of KG) *For any instance I and any π-free expression E*

$$\forall t \in QA(I, Env(E,F)).\forall t' \in TC(E,t).t' \in KG(E,t) \Rightarrow t' \in I.$$

We prove this results together with the following claim.

Theorem 4.23 (Completness of KG for $\{\sigma, \times\}$-queries) *For any instance I and any $\{\sigma,\times\}$-expression E*

$$\forall t \in QA(I, Env(E,F)).\forall t' \in TC(E,t).t' \in KG(E,t) \Leftrightarrow t' \in I.$$

Proof[of Theorems 4.22 and 4.23] We start by proving the completeness of $\{\sigma, \times\}$-expressions using induction over the structure of E.

- Let $E = R$ for some relation name R. Take any $t \in QA(I, H(R, F))$ and observe that $TC(R, t)$ contains only one element $R(t)$. We also note that $t \in QA(I, H(R, F))$ implies $R(t) \in H(I, F)$ and consequently $R(t)$ belongs either to $C(I, F)$ or $S(I, F)$. Recall that the active conflict hypergraph $G^{\mathcal{A}}(I, F)$ contains only tuples from $S(I, F)$ and stores the information if a tuple from $S(I, F)$ is base (i.e., belongs to I). Therefore $R(t)$ belongs to I if and only if $R(t)$ is not a non-base tuple of $G^{\mathcal{A}}(I, F)$, i.e. $R(t) \in KG(R, t)$.
- Consider the expression $\sigma_\rho(E)$, take any $t \in QA(I, Env(\sigma_\rho(E), F))$, and observe that $t \in QA(I, Env(E, F))$. For any $t' \in TC(\sigma_\rho(E), t) = TC(E, t)$ we have $t' \in KG(E, t) = KG(\sigma_\rho(E), t)$ if and only if $t' \in I$ by induction hypothesis.
- Consider the expression $E_1 \times E_2$, take any $t \in QA(I, Env(E_1 \times E_2, F))$, and note that $t = t_1 \cdot t_2$ and $t_i \in QA(I, Env(E_i, F))$ for $i \in \{1, 2\}$. Now, take any $t' \in TC(E_1 \times E_2, t) = TC(E_1, t_1) \cup T(E_2, t_2)$. For the *only if* part, suppose that $t' \in KG(E, t)$. This implies that $t' \in KG(E_i, t_i)$ for some $i \in \{1, 2\}$. By Fact 4.21 $t' \in TC(E_i, t_i)$ and by induction hypothesis $t' \in I$. For the *if* part, suppose $t' \in I$. Because $t' \in TC(E_i, t_i)$ for some $i \in \{1, 2\}$ by induction hypothesis $t' \in KG(E_i, t_i)$ and consequently $t' \in KG(E, t)$.

Next, we extend the inductive argument above to show soundness for expressions that use also the difference and union operators.

- Consider the expression $E = E_1 \cup E_2$, take any $t \in QA(I, Env(E, F))$, and note that $t \in QA(I, Env(E_i, F))$ for some $i \in \{1, 2\}$. Now, take any $t' \in KG(E, t)$ and suppose that $t' \in KG(E, t)$. This implies that $t' \in KG(E_1, t)$, by Fact 4.21 we have that $t' \in KG(E_1, t)$, and by induction hypothesis $t' \in I$.
- Consider the expression $E = E_1 \setminus E_2$, take any $t \in QA(I, Env(E, F))$, and note that $t \in QA(I, Env(E_1, F))$. Now, take any $t' \in TC(E, t)$ and suppose $t' \in KG(E, t) = KG(E_1, t)$. Then, by induction hypothesis $t \in I$.

□

4.5 Extended enveloping

Soundness of knowledge gathering, although helpful, is not sufficient to allow the system to be scalable to queries whose envelopes are large. Executing even one membership query for every tuple maybe very cost prohibitive. To further improve the process of knowledge gathering we propose to extend the envelope expression so that the resulting tuple carry enough information to derive *all relevant facts*. We illustrate this approach in the following example.

Example 4.24 (cont. of Example 4.19) For the previously considered expression $E = \sigma_{1=a}(R \times (R \cup P))$ the extended envelope is $\sigma_{1=a}(R \times (R \cup P)) \xleftarrow{3,4} R \xleftarrow{3,4} P$, where $\leftarrow$ is the left outer join operator[1]. Suppose now, $I = \{R(a,b), R(e,f), P(c,d), P(e,f)\}$. Then the evaluation of the extended envelope expression yields the following:

$\sigma_{1=a}(R \times (R \cup P)) \xleftarrow{3,4} R \xleftarrow{3,4} P$							
a	b	a	b	a	b	$\perp$	$\perp$
a	b	c	d	$\perp$	$\perp$	c	d
a	b	e	f	e	f	e	f

Now, consider the tuple $(a,b,c,d,\perp,\perp,c,d)$. We can decompose it into two parts (a,b,c,d) and $(\perp,\perp,c,d)$. The first part is simply the tuple from the envelope $Env(E,F)$, and it can be used to infer the fact $R(a,b)$ as described in the previous example. The second part allows us to make two other important inferences. Namely, $R(c,d) \notin I$ and $P(c,d) \in I$.

Our goal is to minimally extend the envelope expression with outer join expression so that we can derive all relevant facts. To identify what information cannot be gathered from evaluating the standard envelope expression we generalize the definition TC and KG to *dummy* tuples consisting of distinct variables. For TC the definition remains the same. The definition of KG needs to be augmented for the base case $KG(R,\bar{x})$. Because in this case

[1]For clarity we simplify the notion of the outer join condition. When writing $S \xleftarrow{3,4} T$ we mean $S \xleftarrow{S.3=T.1 \wedge S.4=T.2} T$, and we assume the left join operator is left associative

knowledge gathering is complete, we identify the fact that the information is derived with the following rule

$$KG(R, \bar{x}) = \{R(\bar{x})\}.$$

We note that for dummy tuples we do not need $G^{\mathcal{A}}(I, F)$ to evaluate KG. Finally, we observe that because neither TC nor KG operator permutes the elements of the tuples, they result in a set of atomic formulas having always consecutive elements of the input tuple.

Fact 4.25 *For any E and any sequence of variables and constants $\bar{v} = (v_1, \ldots, v_{|E|})$, if u is an element of $TC(E, \bar{v})$ or $KG(E, \bar{v})$, then $u = R(v_i, \ldots, v_{i+|R|-1})$ for some $R \in \mathcal{S}$ and some $i \in \{1, \ldots, |E|\}$. Moreover, $KG(E, \bar{x}) \subseteq TC(E, \bar{x})$.*

Fact 4.26 *For any E and any t compatible with E*

$$TC(E, t) = \{R(t_i, \ldots, t_{i+|R|-1}) | R(x_i, \ldots, x_{i+|R|-1}) \in TC(E, \bar{x})\},$$

where $\bar{x} = (x_1, \ldots, x_{|E|})$.

Fact 4.27 *For any instance I, any π-free E, and any $t \in QA(I, Env(E, F))$*

$$\begin{aligned}KG(E, t) = \{R(t_i, \ldots, t_{i+|R|-1}) | R(x_i, \ldots, x_{i+|R|-1} \in TC(E, \bar{x}) \quad \textit{and} \\ (t_i, \ldots, t_{i+|R|-1}) \textit{ is not a non-base tuple of } G^{\mathcal{A}}(I, F),\end{aligned}$$

where $\bar{x} = (x_1, \ldots, x_{|E|})$.

Example 4.28 (cont. of Example 4.24) For the expression $E = \sigma_{1=a}(R \times (R \cup P))$ and $\bar{x} = (x_1, x_2, x_3, x_4)$ we have:

$$\begin{aligned}TC(E, \bar{x}) &= \{R(x_1, x_2), P(x_3, x_4), R(x_3, x_4)\}, \\ KG(E, \bar{x}) &= \{R(x_1, x_2)\}.\end{aligned}$$

$R(x_1, x_2) \in TC(E)$ means that for any tuple $t = (t_1, t_2, t_3, t_4)$ from the evaluation of the envelope expression for E, *Prover* may perform the tuple check $R(t_1, t_2)$. We have also $R(x_1, x_2) \in KG(E)$ and therefore we are able to answer this check using knowledge gathering. On the other hand $R(x_3, x_4) \in TC(E)$ means that for t *Prover* may also perform a tuple check $R(t_3, t_4)$. However, $R(x_3, x_4) \notin KG(E)$, which means that we can answer the tuple check $R(t_3, t_4)$ without executing a membership query on the database, even though we are able to answer the tuple check $R(t_1, t_2)$.

Similar examples can be used to show that the simple knowledge gathering is not sufficient to avoid membership checks when processing expressions with the difference operator.

Now, we observe that KG on dummy tuples identifies the tuple checks that can be always answered from the standard envelope expression.

Lemma 4.29 *For any instance I, any $t \in QA(I, Env(E, I))$, and any $R(t_i, \ldots, t_{i+|R|-1}) \in TC(E, t)$ if $R(x_i, \ldots, x_{i+|R|-1}) \in KG(E, (x_1, \ldots, x_{|E|}))$, then $R(t_i, \ldots, t_{i+|R|-1}) \in I \Leftrightarrow R(t_i, \ldots, t_{i+|R|-1}) \in KG(E, t)$.*

Proof By induction over the structure of E. We note that the cases of R, $\sigma_\rho(E)$, and $E_1 \times E_2$ are proved by Theorem 4.23 and Fact 4.27. For the cases of $E_1 \cup E_2$ and $E_1 \setminus E_2$ we need to show only the $\Rightarrow$ implication; the implication in the opposite direction follows from Theorem 4.22 and Fact 4.27.

Consider the expression $E = E_1 \cup E_2$, take any $t \in QA(I, Env(E, F))$, and any $R(t_i, \ldots, t_{i+|R|-1}) \in TC(E, t)$ such that $R(\bar{x}') \in KG(E, \bar{x})$, where $\bar{x} = (x_1, \ldots, x_{|E|})$ and $\bar{x}' = (x_i, \ldots, x_{i+|R|-1})$. We observe that $R(x_i, \ldots, x_{i+|R|-1})$ belongs to both $KG(E_1, (x_1, \ldots, x_{|E_1|}))$ and $KG(E_2, (x_{|E_1|+1}, \ldots, x_{|E_1|+|E_2|}))$. Then regardless if $R(t_i, \ldots, t_{i+|R|-1})$ belongs to $TC(E_1, t)$ or $TC(E_2, t)$, by induction hypothesis if $R(t_i, \ldots, t_{i+|R|-1}) \in I$ then $R(t_i, \ldots, t_{i+|R|-1}) \in KG(E, t)$. The case $E = E_1 \setminus E_2$ is proved analogously. $\Box$

Definition 4.30 (Complementary set) For a given π-free expression E, the *complementary set* $\Gamma(E)$ is defined as follows:

$$\Gamma(E) = TC(E, \bar{x}) \setminus KG(E, \bar{x}),$$

where $\bar{x} = (x_1, \ldots, x_{|E|})$.

Example 4.31 (cont. of Example 4.28) The complementary set for the expression $E = \sigma_{1=a}(R \times (R \cup P))$ is:

$$\Gamma(E) = \{R(x_3, x_4), P(x_3, x_4)\}.$$

Next, we show how to use the complementary set to obtain an extended envelope expression allowing to infer all relevant facts.

Definition 4.32 (Extended envelope expression) For a π-free expression the *extended envelope* $ExtEnv(E, F)$ is defined as

$$ExtEnv(E,F) = Env(E,F) \xleftarrow[R(x_i,\ldots,x_{i+|R|-1}) \in \Gamma(E)]{\bigwedge_{j=1}^{|R|} E.(i+j-1)=R.j} R.$$

The notation means that we have as many outer joins as there are elements in $\Gamma(E)$. They can appear in any order. To facilitate knowledge gathering we define an auxiliary expression $Aux(E)$ of a signature compatible with $ExtEnv(E, F)$:

$$Aux(E) = E \times \bigtimes_{R(x_i,\ldots,x_{i+|R|-1}) \in \Gamma(E)} R.$$

For both $ExtEnv(E, F)$ and $Aux(E)$ the elements of $\Gamma(E)$ need to be considered in the same order.

Because we assume the set semantics for our database instance and use π-free expressions, using outer joins results in a one-to-one correspondence between the tuples of the envelope and the tuples of extended envelope.

Fact 4.33 *For a given instance I and a π-free expression E, the map $t \mapsto t[1, \ldots, |E|]$ is a one-to-one map of $QA(I, ExtEnv(E, F))$ onto $QA(I, Env(E, F))$.*

To infer information about the database instance state based on the tuples of extended envelope we need to further extend knowledge gathering to handle *null* values:

$$KG(R, (\perp, \ldots, \perp)) = \emptyset.$$

Then by Lemma 4.29 and Fact 4.33 we get the following.

Corollary 4.34 (Soundness and completeness of extended knowledge gathering) *For any instance I and any π-free expression E*

$$\forall t \in QA(I, ExtEnv(E, F)).\forall t' \in TC(E, t[1, \ldots, |E|]).t' \in KG(Aux(E), t) \Leftrightarrow t' \in I.$$

4.6 Experimental evaluation

4.6.1 The setting for the experiments

In this section we present results of an experimental evaluation of the Hippo system. The system has been implemented to handle denial constraints only. Among available methods for computing consistent query answers, only the query rewriting technique [ABC99] seems to be feasible for large databases. This is why in this work we compare the following engines:

SQL An engine that executes the given query on the underlying RDBMS, and returns the query result. This method doesn't return consistent query answers, but provides a baseline to observe the overhead of computing consistent query answers using the proposed methods.

QR Using the SQL engine, we execute the rewritten query constructed as described in [ABC99]. More details on this approach can be found in Section 4.7.

KG This method constructs the basic envelope expression and uses knowledge gathering (Section 4.4).

ExtKG This engine constructs the extended envelope expression (Section 4.5).

Generating test data

Every test was performed with the database containing two tables P and Q, both having three attributes X, Y, Z. For the constraints, we took a functional dependency $X \rightarrow Z$ in each table. The test databases had the following parameters:

- n : the number of base tuples in each table,
- m : the number of additional conflicting tuples.

The content of the tables is generated in the following way:

1. Insert n different base tuples with X and Z being equal and taking subsequent values $0, \ldots, n-1$, and Y being randomly drawn from the set $\{0, 1\}$.

2. Insert m different conflicting tuples with X taking subsequent values $\{0, \lceil n/m \rceil, \lceil 2 * n/m \rceil, \ldots, \lceil (m-1) * n/m \rceil\}$, $Z = X + 1$, and Y being randomly drawn from the set $\{0, 1\}$.

In addition, we materialize the core expressions $C(R, F)$ and $C(P, F)$ in auxiliary tables P_{core} and Q_{core}.

Example 4.35 For $n = 4$ and $m = 2$ the contents of P can be generated as follows.

1. First we insert the base tuples $(0, 1, 0), (1, 0, 1), (2, 0, 2)$, and $(3, 1, 3)$ into P.

2. Then we insert the following conflicting tuples $(0, 1, 1)$ and $(2, 0, 3)$ into P.

3. P_{core} contains only the tuples $(1, 0, 1)$ and $(3, 1, 3)$.

In every table constructed in the such a manner the number of tuples is $n + m$, and the number of conflicts is m.

The environment

The implementation is done in Java2, using PostgreSQL (version 7.3.3) as the relational back-end. All test have been performed on a PC with a 1.4GHz AMD Athlon processor under SuSE Linux 8.2 (kernel ver. 2.4.20) using Sun JVM 1.4.1.

4.6.2 Test results

Testing a query with a given engine consisted of computing the consistent[2] answers to the query and then iterating over the results. Iteration over the result is necessary, as the subsequent elements of the consistent query answer set are computed by Hippo in a lazy manner (this allows us to process results bigger than available main memory). Every test has been repeated three times and the median taken. Finally, we note that the cost of

[2]Except when using **SQL** engine.

computing the conflict hypergraph, which is incurred only once per session, is ignored while estimating the time of the query evaluation. To identify the time required for hypergraph construction we performed separate experiments (presented at the end of this section).

Simple queries

We first compared performance of the different engines on simple queries: join, union, and difference. Because we performed the tests for large databases, we added a range selection to the given query to obtain small query results, factoring out the time necessary to write the outputs. As parameters in the experiments, we considered the database size, the conflict percentage, and the estimated result size.

Figure 4.2 shows the execution time for join as a function of the size of the database. In the case of $\{\sigma, \times\}$-expressions (thus also joins), the execution times of **KG**, **ExtKG** and **SQL** are essentially identical. Since no membership queries have to be performed, it means that for simple queries the work done by *Prover* for all tuples is practically negligible.

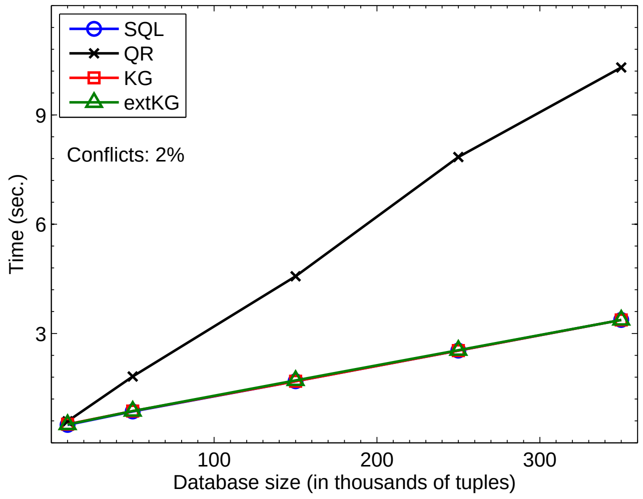


Figure 4.2: Execution time for a join query: $\sigma_{X<200}(P_Join)$.

Figure 4.3 contains the results for union. It shows that basic knowledge gathering **KG** is not sufficient to efficiently handle union. The cost of performing membership queries for all tuples is very large. Note that query rewriting is not applicable to union queries.

Figure 4.4 contains the results for set difference. Because the execution time for **KG**

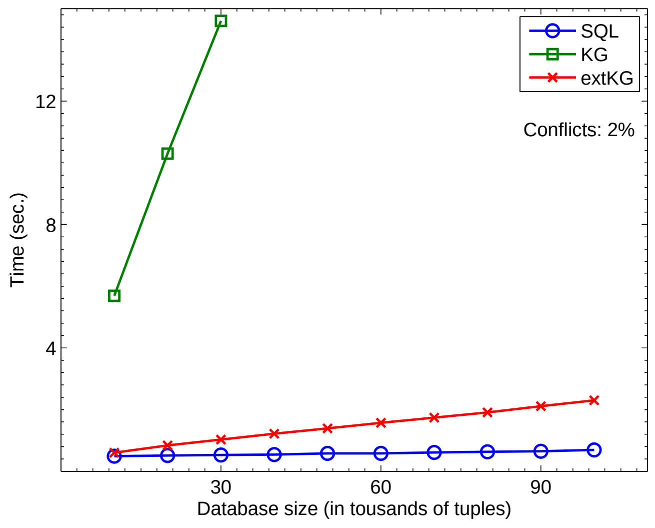


Figure 4.3: Execution time for a union query: $\sigma_{X<200}(P \cup Q)$.

was much larger than values of other solutions, for readability it has not been included on this figure. Here, the execution time is a function of the percentage of conflicts. We note that **ExtKG** performs as well as **QR** and both are approximately twice slower than **SQL**.

Complex queries

In order to estimate the cost of extending the envelope we considered a complex union query

$$\sigma_{X<d}(P \bowtie_X Q \bowtie_X P \bowtie_X Q \cup Q \bowtie_X P \bowtie_X Q \bowtie_X P),$$

with d being a parameter that will allow us to control the number of tuples processed by each engine. To assure no membership queries will be performed, we have to add 8 outer joins. The main goal was to compare two versions of knowledge gathering: **KG** and **ExtKG**. We have also included the results for **SQL**. (It should be noted here that this query has common subexpressions and RDBMS might use this to optimize the query evaluation plan. PostgreSQL, however, does not perform this optimization.)

In Figure 4.5 we see that **KG** outperforms **ExtKG** only in the case when the number of processed tuples is very small. As the result size increases, the execution time of **ExtKG** grows significantly slower than that of **KG**. We notice also that **ExtKG** needs 2–3 times

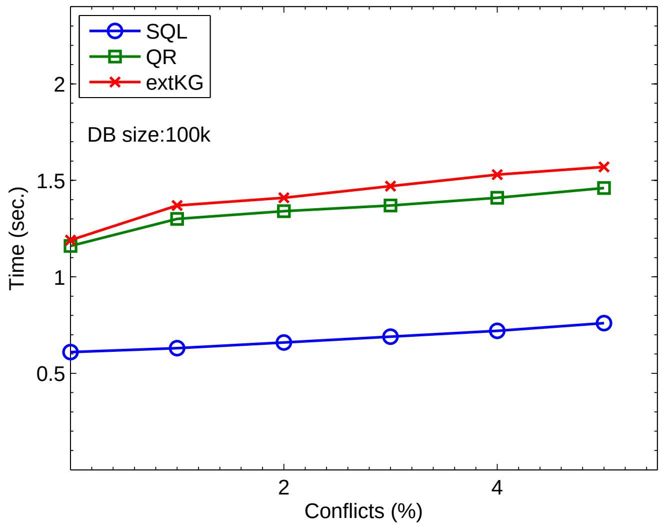


Figure 4.4: Execution time for a difference query $\sigma_{X<200}(P \setminus Q)$.

more time than **SQL** but the execution times of both grow in a similar fashion.

Hypergraph computation

The time of constructing the hypergraph is presented on Figure 4.6). It depends on the total number of conflicts and the size of the database.

It should be noticed here that the time of hypergraph construction consists mainly of the execution time of conflict detection queries. Therefore, the time of hypergraph computation depends also on the number of integrity constraints and their arity.

4.7 Related work

In general, three different approaches to compute consistent query answers have been proposed: query rewriting, logic programming, and compact representation of repairs. Hippo is a system following the last approach using conflict hypergraphs.

4.7.1 Query rewriting

Query rewriting was the first approach proposed to compute consistent query answers [ABC99]. A query Q is rewritten into a query Q' whose evaluation returns the set of consistent query

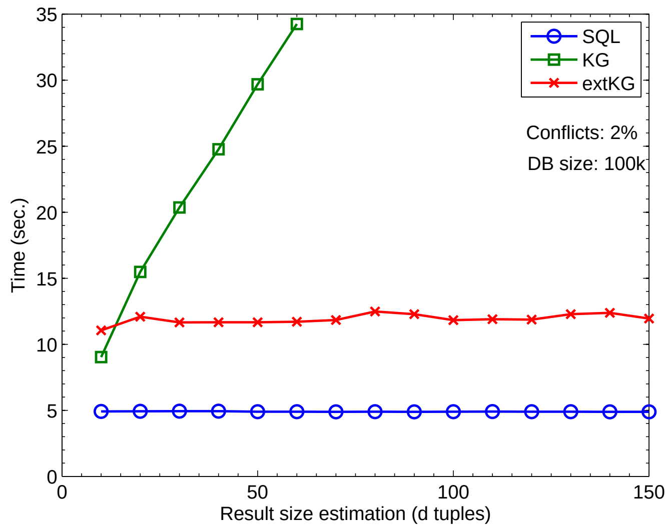


Figure 4.5: Impact of the result size for the query $\sigma_{X<d}(P \bowtie_X Q \bowtie_X P \bowtie_X Q \cup Q \bowtie_X P \bowtie_X Q \bowtie_X P)$.

answers to Q. An indisputable advantage of this approach is the ease of its incorporation into already existing applications. However, applicability of this approach is limited and certain conjunctive queries are known not to have rewritings [CM05].

[ABC99] uses the notion of *residues* obtained from constraints to identify potential impact of integrity violations on the query results. The residues are used to construct rewriting rules for the atoms used in the query. This approach has been shown to be applicable to quantifier-free conjunction of literals in the presence of binary universal constraints.

Example 4.36 Consider schema consisting of two relation names $\mathcal{S} = \{R(A, B), P(C, D)\}$ and let the set of integrity constraints consist of the following two formulas:

$$\forall x, y, y'. \neg R(x, y) \vee \neg R(x, y') \vee y = y', \tag{4.1}$$

corresponding to the functional dependency $R : A \to B$ and

$$\forall x, y. \neg P(x, y) \vee R(x, y) \tag{4.2}$$

corresponding to the full IND $P[C, D] \subseteq R[A, B]$. The rule obtained from the con-

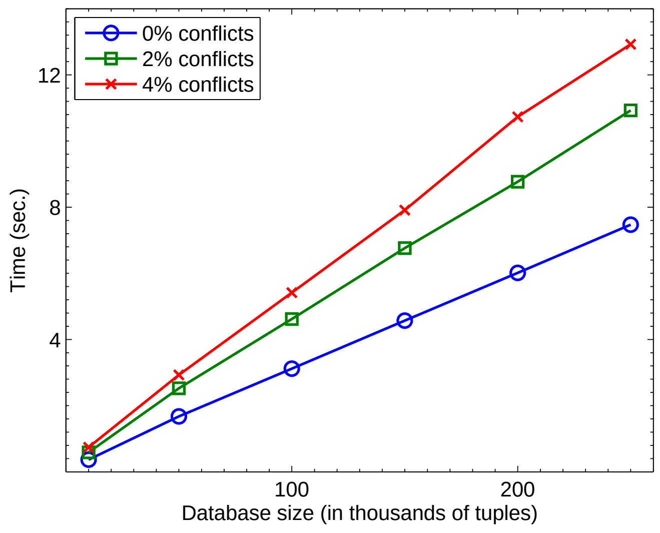


Figure 4.6: Hypergraph computation time.

straint (4.1) is[3]

$$R(x,y) \mapsto R(x,y)\{\forall y'.\neg R(x,y') \lor y = y'\} \tag{4.3}$$

and the rules obtained from the constraint (4.2) are

$$P(x,y) \mapsto P(x,y)\{R(x,y)\}, \tag{4.4}$$

$$\neg R(x,y) \mapsto \neg R(x,y)\{\neg P(x,y)\}. \tag{4.5}$$

Intuitively, the rules specify the conditions assuring that the atom is not affected by integrity violations. For instance the rule (4.4) states that an atom $P(a,b)$ is not involved in violation of the full IND $P[C,D] \subseteq R[A,B]$ if $R(a,b)$ is also present in database. Therefore, if we consider the query $Q = P(x,y)$, it has to be rewritten into the following one

$$P(x,y) \land R(x,y).$$

Because the atom $R(a,b)$ can be involved in violations of the FD $R : A \to B$, the rule (4.3)

[3]Actually for an FD we obtain two rules, one for each literal in the constraint, but the formulas are equivalent.

is used to further augment the query

$$P(x, y) \wedge R(x, y) \wedge (\forall y'.\neg R(x, y') \vee y = y').$$

For similar reasons we apply the rule (4.5) to finally obtain the rewritten query

$$Q' = P(x, y) \wedge R(x, y) \wedge (\forall y'.(\neg R(x, y') \wedge \neg P(x, y')) \vee y = y').$$

Naturally, Q' returns the consistent answers to Q.

Chomicki and Marcinkowski [CM05] observe that if the set of constraints contains one FD per relation only, the conflict graph is a union of disjoint full multipartie graphs. This simple structure allows to construct rewriting for conjunctive queries without repeated relation names and no variable sharing. They also show that relaxing the conditions imposed on the queries and constraints leads to intractability: consistent query answering becomes coNP-complete. This also proves that there exist conjunctive queries for which rewriting does not exists; any first-order query can be evaluated in AC^0 and no coNP-complete problem belongs to this complexity class [Pap94].

The result of Chomicki and Marcinkowski has been further generalized by Fuxman and Miller [FM05] to allow restricted variable sharing (joins) in the conjunctive queries. The class C_{forest} of allowed queries is defined using the notion of *join graph* of a query whose vertices are the literals used in the query and an edge runs from a literal R_i to literal R_j if there is a variable which occurs on a non-key attribute of R_i and any attribute of R_j (both occurrences have to be different if $i = j$). The class C_{forest} consist of queries whose join graph is a forest, the joins are full and the join conditions are non-key to key[4].

In [FFM05] Fuxman et al. present the ConQuer system which computes consistent answers to queries from C_{forest}. The queries can also use aggregates, and then *range*-consistent answers are computed [ABC+03b]: minimal intervals containing the set of values of the aggregate obtained over the repairs. This allows the system to compute consistent answer to 20 out of 22 queries of the TCP-H decision support benchmark. The experimental

[4]Essentially, those are *foreign-key* joins, but we note that the discussed framework does not handle violations of foreign-key constraints.

evaluation of the system shows that the system performs reasonably well and is scalable w.r.t. both the size of the database and the number of conflicts in the database. We observe that the rewritten queries are obtained from the original queries by adding subexpression checking for conflicts. This leads to an overhead; executing the rewritten query takes on average twice the time of executing the original query. However, as shown by our experiments, the overhead can be avoided if the conflicts are detected once and stored in main memory. On the other hand, the rewriting approach is scalable to the number of conflicts, although it is yet to be seen if databases with large numbers of conflicts are common in practice (and if querying them can be informative at all).

Very recently Lembo et al. [LRR06] extended the rewriting for queries from C_{forest} to include union. However, no experimental evaluation has been performed so far.

4.7.2 Logic programs

Several different approaches have been developed to compute consistent query answers using logic programs with disjunction and classical negation [ABC03a, BB03, EFGL03, GGZ01, GGZ03, VNV02]. Essentially, all of them use disjunctive rules to model the process of repairing violations of constraints. In this way stable models of a program corresponds to the repairs of the inconsistent database. A query evaluated under the *cautious* semantics returns the answers present in every model, which naturally yields the consistent query answers.

Example 4.37 Recall the schema from Example 4.36: $\mathcal{S} = \{R(A, B), P(C, D)\}$ with $F = \{R : A \to B, P[C, D] \subseteq R[A, B]\}$. We assume that the database facts, the extensional database, are given with the predicates $P(X, Y)$ and $R(X, Y)$. The *repairing* program defines the repairs using intentional predicates $P'(X, Y)$ and $R'(X, Y)$ resp. The program consists of the following rules:

- *Triggering* rules which specify the possible repairing actions if a constraint violation is detected.

$$\neg P'(X,Y) \vee R'(X,Y) \leftarrow P(X,Y), \textbf{not}\, R(X,Y).$$
$$\neg R'(X,Y) \vee \neg R'(X,Y') \leftarrow R(X,Y), R(X,Y'), Y \neq Y'.$$

- *Stabilizing* rules which ensure that the integrity constraints are satisfied in the constructed repair.

$$R'(X,Y) \leftarrow P'(X,Y).$$
$$\neg P'(X,Y) \leftarrow \neg R'(X,Y).$$
$$\neg R'(X,Y) \leftarrow R'(X,Y'), R(X,Y), Y \neq Y'.$$

- *Persistence* rules which copy facts from the original database to the repair unless the fact has been removed in the repairing process.

$$R'(X,Y) \leftarrow R(X,Y), \textbf{not}\, \neg R'(X,Y).$$
$$P'(X,Y) \leftarrow P(X,Y), \textbf{not}\, \neg P'(X,Y).$$

Assume that the intentional database contains two facts $R(1,1)$ and $P(1,2)$, then the repairing program has two stable models:

$$\mathcal{M}_1 = \{R(1,1), P(1,2), R'(1,1), \neg P'(1,2)\},$$
$$\mathcal{M}_2 = \{R(1,1), P(1,2), R'(1,2), P'(1,2), \neg R'(1,1)\}.$$

Now, if we consider the query $Q(x) = \exists y.R(x,y)$, the corresponding Datalog query is

$$Q'(X) \leftarrow R'(X,Y).$$

The set of answers to this query under the cautious semantics is $\{(1)\}$, which is also the set of consistent answers to $Q(x)$.

The main advantage of using this approach is its generality: typically arbitrary first-order (or even Datalog$^{\neg}$) queries are handled in the presence of universal constraints. Also, the repairing programs can be easily evaluated with exciting logic program environments like Smodels or `dlv` [EFLP00]. We note, however, that the systems computing answers to logic programs usually perform grounding, which may be cost prohibitive if we are to work with large databases. Another disadvantage of this approach is the fact that the class of disjunctive logic programs is known to be Π_p^2-complete.

These difficulties are addressed in the INFOMIX system [EFGL03] with several optimizations geared toward effective execution of repairing programs. One is *localization* of conflicts with identification of the *affected database* which consists of all tuples involved in constraint violations and all syntactically propagated *conflict-bound* tuples (analogous to applying T_F^*). We observe that in the presence of full TPGs and denial constrains the affected database is identical to the *shell instance*. Another optimization involves using bit-vectors to encode tuple membership to each repair and subsequent use of bitwise aggregate function to find tuples present in every repair. This optimization, however, may be insufficient to handle databases with large numbers of conflicts because typically the number of repairs is exponential in the number of conflicts.

Very recently, this deficiency has been addressed with *repair factorization* [EFGL07]. Essentially, the affected database is decomposed into parts that are conflict-disjoint (no two mutually conflicting tuples are in separate parts). When computing consistent answers to a query only parts that are simultaneously spanned by the query are considered at a time. Again, we note the analogy to computing consistent query answers using hypergraphs: when finding whether *true* is the consistent answer to a ground query our algorithm considers only the repairs obtained with enumeration of edges adjacent to the tuples from the query. The presented experimental results validate this approach: the system computes consistent query answers in a reasonable time and is scalable w.r.t. the size of the database and the number of conflicts. Tests with up to 200^{1000} conflicts are reported. We note, however, that while for quantifier-free queries, the number of parts that have to be considered is bounded by the size of the query (and hence can be considered fixed), when handling queries with quantifiers the number of parts can be arbitrarily large. Consequently, the repair factorization strategy may turn out to be unsuccessful. It is yet to be seen if the structure of conflicts in practical scenarios allows to benefit from this optimization.

Chapter 5

Preferences

In this chapter we extend the standard framework of repairs and consistent query answer to include preference information.

We start by introducing priorities to define families of preferred families of repairs in Section 5.1. In Section 5.2 we list desired properties of families of preferred repairs. In Section 5.3 we present three different families of preferred repairs, discuss their mutual relationships, and investigate the computational implications of introducing preferences in the framework of consistent query answers. Section 5.4 contains discussion of related work.

We use the standard relational model and restrict integrity constraints to functional dependencies only. Recall that in the presence of FDs only, the conflict hypergraph becomes a standard graph (edges connect exactly two vertices) and also every repair is a subset of the original instance. Then, for a tuple t, the neighborhood of t, denoted by $n(t)$, is the set of all tuples adjacent to t in the conflict graph.

5.1 Priorities and families of preferred repairs

From now on we assume a fixed instance I with a fixed set of functional dependencies F.

Definition 5.1 (Orientation) An *orientation* of a graph $G = (V, E)$ is a binary relation $\succ \subseteq V \times V$ such that if $v \succ u$ then $\{v, u\} \in E$. An orientation is *acyclic* if it contains no cyclic (directed) path.

To represent the preference information, we use acyclic orientations of some (not necessarily all) edges of the conflict graph. Orientations allow us to express the preferences at the level of single conflicts and acyclicity ensures unambiguity of the preference.

Definition 5.2 (Priority) A *priority* $\succ$ (of I w.r.t. F) is an acyclic orientation of $G(I, F)$. A priority $\succ$ is *total* if for every edge $\{t_1, t_2\}$ of $G(I, F)$ either $t_1 \succ t_2$ or $t_1 \succ t_2$. A *prioritized conflict graph* $G(I, F, \succ)$ is a conflict graph $G(I, F)$ with a priority $\succ$ of I w.r.t. F.

From the point of the user interface it is often more natural to define the priority as some acyclic binary relation on tuples of I and then consider the restriction of the priority relation to the conflicting tuples. Clearly, those approaches are equivalent.

Example 5.3 Recall from Example 1.4 the schema $Mgr(Name, Dept, Salary)$ with two FDs $F = \{Name \rightarrow Dept\,Salary, Dept \rightarrow Name\,Salary\}$ and the user preference for conflict resolution: for a conflict created by two tuples referring to the same person, the user prefers to resolve the conflict by removing the tuple with the smaller salary. The prioritized conflict graph for the priority obtained from this preference is presented in Figure 5.1.

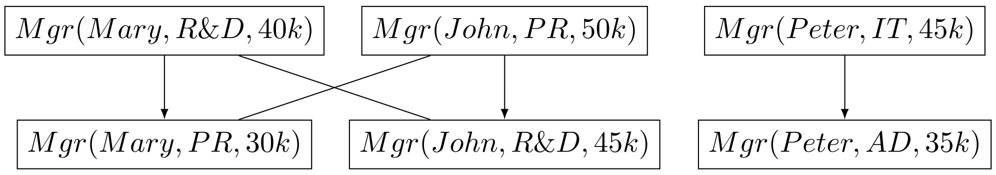


Figure 5.1: A prioritized conflict graph.

We note that in our framework we do not assume any other properties of the priority. In particular, it does not have to be transitive.

Extending an orientation consists of orienting some edges that were not oriented before. Formally, an orientation $\succ'$ is an *extension* of a priority $\succ$ if $\succ'$ is a priority and $\succ' \supseteq \succ$. Note that $\succ'$ is also acyclic and defined only for pairs of tuples that create a conflict.

Definition 5.4 (Preferred repairs) A *family of preferred repairs* is a function $\mathcal{X}Rep$ defined on triplets $(I, F, \succ)$, where $\succ$ is a priority in I w.r.t. a set of FDs F, such that $\mathcal{X}Rep(I, F, \succ) \subseteq Rep(I, F)$. We say that a family $\mathcal{Y}Rep$ *subsumes* a family $\mathcal{X}Rep$, denoted $\mathcal{X}Rep \sqsubseteq \mathcal{Y}Rep$, if for every $(I, F, \succ)$ we have that $\mathcal{X}Rep(I, F, \succ) \subseteq \mathcal{Y}Rep(I, F, \succ)$.

5.1.1 Preferred consistent query answers

We generalize the notion of consistent answers to closed FOL queries (Definition 2.6) by considering only preferred repairs when evaluating a query (instead of all repairs). We can easily generalize our approach to open queries along the lines of Chapter 4. Recall that for a closed query Q *true* is an answer to Q in I if $I \models Q$ in the standard model-theoretic sense.

Definition 5.5 ($\mathcal{X}$-preferred consistent query answer) Given a closed query Q, a triple $(I, F, \succ)$, and a family of preferred repairs $\mathcal{X}Rep$, *true* is the *$\mathcal{X}$-preferred consistent query answer* to Q in I w.r.t. F and $\succ$, written $I \models^{\mathcal{X}}_{F,\succ} Q$, if for every $I' \in \mathcal{X}Rep(I, F, \succ)$ we have $I' \models Q$.

Note that we obtain the original notion of consistent query answer (Definition 2.6) if we consider the family of all repairs $Rep(I, F)$. Also, note that if $\mathcal{Y}Rep \sqsubseteq \mathcal{X}Rep$ then $I \models^{\mathcal{X}}_{F,\succ} Q$ implies $I \models^{\mathcal{Y}}_{F,\succ} Q$.

5.1.2 Data complexity

We also adapt the decision problems to include the priority. Note that the priority relation is of size polynomial in the size of the database instance, and therefore it is natural to make it a part of the input. For a family $\mathcal{X}Rep$ of preferred repairs the decision problems we study are defined as follows:

(i) *$\mathcal{X}$-preferred repair checking* i.e., the complexity of the following set

$$\mathcal{B}^{\mathcal{X}}_{F} = \{(I, \succ, I') : I' \in \mathcal{X}Rep(I, F, \succ)\}.$$

(ii) *$\mathcal{X}$-preferred consistent query answering* i.e., the complexity of the following set

$$\mathcal{D}^{\mathcal{X}}_{F,Q} = \{(I, \succ) : \forall I' \in \mathcal{X}Rep(I, F, \succ).I' \models Q\}.$$

5.2 Properties of families of preferred repairs

In this section we investigate desirable properties of arbitrary families of preferred repairs. We fix an instance I and a set of functional dependencies F.

$\mathcal{P}1$ **Non-emptiness**

Because the set of preferred repairs is used to define preferred consistent query answers, it is important that for any preference information the framework is not trivialized by an empty set of preferred repairs:

$$\mathcal{X}Rep(I, F, \succ) \neq \varnothing.$$

$\mathcal{P}2$ **Monotonicity**

The operation of extending the preference allows to improve the state of our knowledge of the real world. The better such knowledge is the finer the (preferred consistent) answers we should obtain. This is achieved if extending the preference can only narrow the set of preferred repairs:

$$\succ_1 \subseteq \succ_2 \Longrightarrow \mathcal{X}Rep(I, F, \succ_2) \subseteq \mathcal{X}Rep(I, F, \succ_1).$$

$\mathcal{P}3$ **Non-discrimination**

Removing repairs from consideration must be justified by existing preference information. In particular, no repair should be removed if no preference is given:

$$\mathcal{X}Rep(I, F, \varnothing) = Rep(I, F).$$

$\mathcal{P}4$ **Categoricity**

Ideally, a preference that cannot be further extended (priority is total) should specify how to resolve every conflict:

$$\succ \text{ is total} \Longrightarrow |\mathcal{X}Rep(I, F, \succ)| = 1.$$

$\mathcal{P}5$ **Conservativeness**

We also note that properties $\mathcal{P}2$ and $\mathcal{P}3$ imply that preferred repairs are a subset of all repairs:

$$\mathcal{X}Rep(I, F, \succ) \subseteq Rep(I, F).$$

5.3 Prioritized repairing

In this section we investigate several different families of preferred repairs. Again, I denotes a fixed instance and F a set of functional dependencies.

5.3.1 Globally optimal repairs

We start from investigating a notion of repair optimality inspired by work done in preferred models of logic programs [VNV02] and preferential reasoning [Hal97].

Definition 5.6 (Globally optimal repairs $\mathcal{G}Rep$) A repair I' is *globally optimal* if no nonempty subset X of tuples from I' can be replaced with a nonempty set Y of tuples from I such that

$$\forall x \in X. \exists y \in Y. y \succ x$$

and the resulting set of tuples is consistent. $\mathcal{G}Rep(I, F, \succ)$ is the set of all globally optimal repairs of I.

The notion of global optimality identifies repairs whose compliance with the priority cannot be further improved; In Example 1.4 only I_1 is globally optimal. Before investigating the properties of $\mathcal{G}Rep$ we present an alternative characterization of globally optimal repairs.

Proposition 5.7 *For a given priority $\succ$ and two different repairs I_1 and I_2, we say that I_1* dominates *I_2, denoted $I_1 \gg I_2$, if*

$$\forall x \in I_2 \setminus I_1. \exists y \in I_1 \setminus I_2. y \succ x.$$

The following facts hold:

1. *a repair is globally optimal if and only if it is $\gg$-maximal;*

2. *if $\succ$ is acyclic, then so is $\gg$.*

Proof *(1)* For the *if* part take any $\gg$-maximal repair I', any nonempty set X of tuples from I', and any nonempty set Y of tuples from I such that $(I' \setminus X) \cup Y$ is consistent. Take $Y' = Y \setminus I'$ and note that by $<_I$-minimality of I' for every $y \in Y'$ there exists $x \in I'$ conflicting with y. Moreover, all tuples conflicting with any $y \in Y$ must be contained in the set X or $(I' \setminus X) \cup Y$ would not be consistent. Therefore, any repair I'' containing $(I' \setminus X) \cup Y$ is disjoint with X. This implies that $X = I' \setminus I''$ and $Y' \subseteq I'' \setminus I'$. Finally, by $\gg$-maximality of I' ($I'' \not\gg I'$) we observe that $\forall x \in X. \exists y \in Y'. y \succ x$ cannot be true. Because none of the tuples from $Y \cap I'$ conflicts with any tuple from I', this also implies that I' is globally optimal.

For the *only if* part take any globally optimal repair I' and any repair I''. If we consider $X = I' \setminus I''$ and $Y = I'' \setminus I'$, then global optimality of I' implies that $I'' \not\gg I'$.

(2) Suppose $\gg$ is cyclic. Then there exists an infinite sequence of repairs (with repetitions)

$$\ldots \gg I_n \gg \ldots \gg I_2 \gg I_1.$$

Naturally, $I_i \setminus I_{i+1}$ is nonempty for any $i \leq 1$. Consider an infinite sequence of tuples $t_1, t_2, \ldots$ defined as follows:

- t_1 is any tuple from $I_1 \setminus I_2$,

- t_{i+1} is any tuple from $I_{i+1} \setminus I_i$ such that $t_{i+1} \succ t_i$ (there exists at least one by $I_{i+1} \gg I_i$).

Because I has only finitely many elements, the sequence $(t_i)_{i=1}^{\infty}$ must have repetitions and so $\succ$ is cyclic; a contradiction. □

Proposition 5.8 *$\mathcal{G}Rep$ satisfies the properties $\mathcal{P}1$-$\mathcal{P}4$.*

The proof of Propositions 5.8, 5.16, and 5.21 can be found in the appendix.

Theorem 5.9 *$\mathcal{G}$-preferred repair checking is coNP-complete and $\mathcal{G}$-preferred consistent query answering is Π_2^p-complete.*

Proof The membership of $\mathcal{G}$-preferred repair checking to coNP follows from Proposition 5.7: to prove that an instance I' is not globally optimal it enough to guess a consistent instance I'' such that $I'' \gg I'$ and I'' is repair (which can be checked in polynomial time [CM05]). Similarly, the membership of $\mathcal{G}$-preferred consistent query answering in Π_2^p follows from Definition 5.5: *true* is not the $\mathcal{G}$-preferred consistent answer to a query if the query is not *true* in some globally optimal repair.

We show Π_2^p-hardness of $\mathcal{D}_{F,Q}^{\mathcal{G}}$ by reducing the satisfaction of $\forall^*\exists^*$QBF formulas to $\mathcal{D}_{F,Q}^{\mathcal{G}}$. Consider the following formula:

$$\psi = \forall x_1, \ldots, x_n . \exists x_{n+1}, \ldots, x_{n+m} . \phi, \tag{5.1}$$

where ϕ is quantifier-free and is in 3CNF, i.e ϕ equals to $c_1 \wedge \ldots \wedge c_s$, and c_k are clauses of three literals $l_{k,1} \vee l_{k,2} \vee l_{k,3}$. We call the variables $x_1, \ldots, x_n$ *universal* and $x_{n+1}, \ldots, x_{n+m}$ *existential.* We use the function q to identify the type of a variable with a given index: $q(i) = 1$ for $i \leq n$ and $q(i) = 0$ for $i > n$. A *valuation* is a (possibly partial) function assigning a boolean value to the variables.

We construct an instance I_ψ over the schema with a single relation

$$R(A_1, B_1, A_2, B_2, A_3, B_3, A_4, B_4)$$

and an acyclic priority $\succ_\psi$. The set of integrity constraints is $F = \{A_1 \rightarrow B_1, A_2 \rightarrow B_2, A_3 \rightarrow B_3, A_4 \rightarrow B_4\}$.

We define the following two auxiliary functions *var* and *sgn* on literals of ϕ:

$$var(x_i) = var(\neg x_i) = i, \qquad sgn(x_i) = 1, \qquad sgn(\neg x_i) = -1.$$

The reduction uses the following types of tuples:

- v_i and $\bar{v}_i$ corresponding to the positive and negative valuations of x_i resp.

$$v_i = R(0, q(i), i, 1, i, 1, i, 1), \qquad \bar{v}_i = R(0, q(i), i, -1, i, -1, i, -1),$$

- d_k corresponding to c_k

$$d_k = R(0, 1, var(l_{k,1}), sgn(l_{k,1}), var(l_{k,2}), sgn(l_{k,2}), var(l_{k,3}), sgn(l_{k,3})),$$

- $t_\exists$ and $t_\forall$ used to partition the set of all repairs

$$t_\exists = R(0, 0, 0, 0, 0, 0, 0, 0), \qquad t_\forall = R(0, 1, 0, 0, 0, 0, 0, 0).$$

For the ease of reference by $L_{k,p}$ we denote the tuple corresponding to the satisfying valuation of literal $l_{k,p}$, i.e.:

$$L_{k,p} = \begin{cases} v_i & \text{when } l_{k,p} = x_i, \\ \bar{v}_i & \text{when } l_{k,p} = \neg x_i. \end{cases}$$

The constructed instance is

$$I_\psi = \{v_1, \bar{v}_1, \ldots, v_{n+m}, \bar{v}_{n+m}, d_1, \ldots, d_s, t_\forall, t_\exists\}.$$

The priority relation $\succ_\psi$ is the unique minimal relation satisfying the following conditions:

$$\begin{aligned}
&v_i \succ_\psi d_k, && \text{if } c_k \text{ uses a positive literal } x_i,\\
&\bar{v}_i \succ_\psi d_k, && \text{if } c_k \text{ uses a negative literal } \neg x_i,\\
&t_\exists \succ_\psi v_i, && \text{for all } i \in \{1, \ldots, n\},\\
&t_\exists \succ_\psi \bar{v}_i, && \text{for all } i \in \{1, \ldots, n\},\\
&t_\exists \succ_\psi t_\forall.
\end{aligned}$$

The query used in the reduction is $Q = t_\exists$.

In Figure 5.2 illustrates the reduction on the following formula:

$$\psi = \forall x_1, x_2, x_3. \exists x_4, x_5. (\neg x_1 \vee x_4 \vee x_2) \wedge (\neg x_2 \vee \neg x_5 \vee \neg x_3).$$

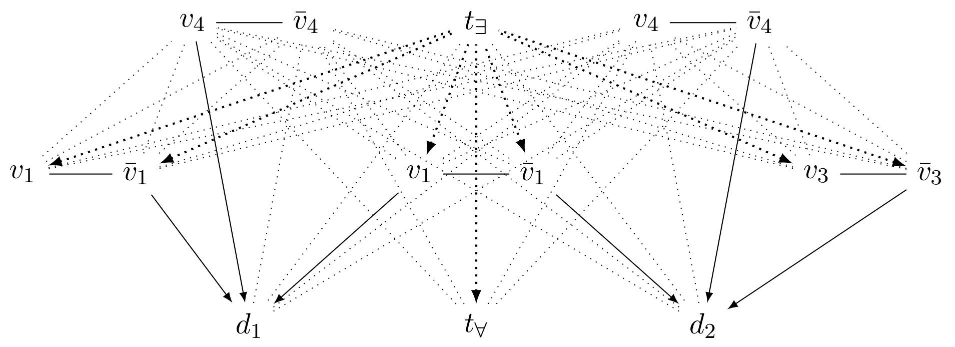


Figure 5.2: The prioritized conflict graph for ψ. Dotted lines used to show the conflicts created with $A_1 \rightarrow B_1$.

We partition the set of all repairs of I_ψ into two disjoint classes: $\exists$-*repairs* that contain $t_\exists$ and $\forall$-*repairs* that do not contain $t_\exists$. We note that because of the FD $A_1 \rightarrow B_1$ every $\forall$-repair contains $t_\forall$. For the same reason, an $\forall$-repair is a subset of $\{v_1, \bar{v}_1, \ldots, v_n, \bar{v}_n, d_1, \ldots, d_n, t_\forall\}$ and an $\exists$-repair is a subset of $\{v_{n+1}, \bar{v}_{n+1}, \ldots, v_{n+m}, \bar{v}_{n+m}, t_\exists\}$.

We use $\exists$- and $\forall$-repairs to represent all possible valuation of existential and universal variables respectively. To easily move from a partial valuation of variables to a repair we define the following two operations:

$$\begin{aligned} I_\exists[V] =& \{v_i | V(x_i) = \mathit{true} \wedge q(i) = 0\} \cup \{\bar{v}_i | V(x_i) = \mathit{false} \wedge q(i) = 0\} \cup \{t_\exists\}, \\ I_\forall[V] =& \{v_i | V(x_i) = \mathit{true} \wedge q(i) = 1\} \cup \{\bar{v}_i | V(x_i) = \mathit{false} \wedge q(i) = 1\} \cup \{t_\forall\} \cup \\ & \left\{ d_k \left| \begin{array}{l} \text{if for every literal } l_{k,i} \text{ of } c_k \text{ s.t. } V \text{ is defined on} \\ \text{the variable used in } l_{k,i} \text{ we have } V \not\models l_{k,i} \end{array} \right. \right\}. \end{aligned}$$

To move in the opposite direction, from a repair to a (possibly partial) valuation we use:

$$V[I'](x_i) = \begin{cases} \mathit{true} & \text{if } v_i \in I', \\ \mathit{false} & \text{if } \bar{v}_i \in I', \\ \mathit{undefined} & \text{otherwise.} \end{cases}$$

We observe that $V[\cdot]$ defines a one-to-one correspondence between $\exists$-repairs and total valuations of existential variables. A similar statement, however, does not hold for $\forall$-repairs, as for some $\forall$-repair I' $V[I']$ may be only a partial valuation of universal variables. To address this deficiency we identify *strict* $\forall$-repairs; a $\forall$-repair I' if and only if $V[I']$ is a total valuation universal variables. Now, $V[\cdot]$ defines a one-to-one correspondence between strict $\forall$-repairs and total valuation of the universal variables. The following result allows us to remove non-strict $\forall$-repairs from consideration.

Lemma 5.10 *Strict $\forall$-repairs are $\gg$-maximal $\forall$-repairs and vice versa.*

Proof First, we prove that $\gg$-maximal $\forall$-repairs are strict. For that we show how for any non-strict $\forall$-repair I' construct a strict $\forall$-repair I'' such that $I'' \gg I'$. Take the partial valuation $V' = V[I']$ and extend it to a total valuation V'' of universal variables by assigning *false* value to variables undefined by V', i.e.

$$V'' = V' \cup \{(x_i, false) | 1 \leq i \leq n \land V'(x_i) \text{ is undefined}\}.$$

We take $I'' = I_\forall[V'']$ and show that

$$\forall t' \in I' \setminus I''. \exists t'' \in I'' \setminus I'. t'' \succ t'.$$

Because I' is an $\forall$-repair there are 3 cases of values of t' to consider:

1. $t' = t_\forall$ is not possible because both I' and I'' are $\forall$-repairs.

2. $t' = v_i$ or $t' = \bar{v}_i$ for some $i \in \{1, \ldots, n\}$ is also impossible because from the construction of I'' we know that

$$I'' \cap \{v_1, \bar{v}_1, \ldots, v_n, \bar{v}_n\} \subseteq I' \cap \{v_1, \bar{v}_1, \ldots, v_n, \bar{v}_n\}.$$

3. $t' = d_k$ for some $k \in \{1, \ldots, s\}$. The neighborhood of t' consists of tuples $t_\exists$, $L_{k,1}$, $L_{k,2}$, and $L_{k,3}$. We observe that none of the tuples belongs to I'. However, one of the tuples must belong to I'' because $t' \notin I''$ (by $<_{I_\phi}$-minimality). Since I'' is an $\forall$-repair, $t_\exists$ does not belong to I'' and therefore for some $p \in \{1, 2, 3\}$ the tuple $L_{k,p}$ belongs to I''. Finally, $t'' = L_{k,p} \succ_\phi t'$.

Now, we show that every strict $\forall$-repair is also $\gg$-maximal among $\forall$-repairs. Suppose otherwise , i.e. for some strict $\forall$-repair I' there exists an $\forall$-repair I'' such that $I' \gg I''$. Since

I' is strict it contains v_i or $\bar{v}_i$ for every $i \in \{1, \ldots, n\}$. By the construction of the priority $\succ_\phi$ the repairs I' and I'' must agree on tuples $v_1, \bar{v}_1, \ldots, v_n, \bar{v}_n$. Therefore $I' = I_\forall[V[I'']]$ and using the reasoning from the previous part we can show that $I'' \gg I'$. Since $\succ_\psi$ is acyclic, this implies that $I' = I''$.

The central result in our reduction follows.

Lemma 5.11 *For any total valuation V $I_\exists[V] \gg I_\forall[V]$ if and only if $V \models \phi$.*

Proof For the *if* part we note that because any $\forall$-repair is disjoint with any $\exists$-repair, it is enough to show that for any tuple $t' \in I_\forall[V]$ there exists a tuple $t'' \in I_\exists[V]$ such that $t'' \succ t'$. For any of the tuples $t_\forall, v_1, \bar{v}_1, \ldots, v_n, \bar{v}_n$ we simply choose $t_\exists$. If d_k belongs to $I_\forall[V]$, we note that none of the neighbors of d_k belongs to $I_\forall[V]$. This implies that none of the literals using a universal variable is satisfied by V. Hence there must exist a literal $l_{k,p}$ (using an existential variable) that is satisfied by V. Consequently, we have $L_{k,p} \in I_\exists[V]$ and $L_{k,p} \succ d_k$.

For the *only if* part for any $k \in \{1, \ldots, s\}$ we consider the conjunct $c_k = l_{k,1} \vee l_{k,2} \vee l_{k,3}$. If none of the literals which use universal variables is satisfied by V, then none of the corresponding $L_{k,p}$ belongs to $I_\forall[V]$ and consequently d_k is in $I_\forall[V]$. Then $I_\exists[V]$ must contain a tuple $L_{k,p'}$ corresponding to one of the literals of c_k using an existential variable. This implies that $V \models l_{k,p'}$ and consequently $V \models c_k$.

This gives us.

Fact 5.12 *The QBF ψ is true if and only if for any strict $\forall$-repair I' there exists a $\exists$-repair I'' such that $I'' \gg I'$.*

Because only a $\exists$-repair can dominate a strict $\forall$-repair and every non-strict $\forall$-repair is dominated by a strict one, we can make a more general statement.

Fact 5.13 *The QBF ψ is true if and only if for any $\forall$-repair I' there exists a repair I'' such that $I'' \gg I'$.*

$\forall$-repairs are defined as repairs that do not contain the tuple $t_\exists$ and thus:

$$\models \psi \iff \models \forall x_1, \dots, x_n . \exists x_{n+1}, \dots, x_{n+m} . \phi \iff$$
$$\forall I' \in \mathit{Rep}(I_\psi, F). I' \models \neg R(t_\exists) \Rightarrow \exists I'' \in \mathit{Rep}(I_\psi, F). I'' \gg I' \iff$$
$$\forall I' \in \mathit{Rep}(I_\psi, F). [\neg \exists I'' \in \mathit{Rep}(I_\psi, F). I'' \gg I'] \Rightarrow I' \models R(t_\exists) \iff$$
$$\forall I' \in \mathcal{G}\mathit{Rep}(I_\psi, F, \succ_\psi). I' \models R(t_\exists) \iff (I_\psi, \succ_\psi) \in \mathcal{D}^{\mathcal{G}}_{F, R(t_\exists)}.$$

Corollary 5.14 *QBF ψ is* true *if and only if* true *is the $\mathcal{G}$-preferred consistent answer to $R(t_\exists)$ in I_ψ w.r.t. F and $\succ_\psi$.*

To show coNP-hardness of $\mathcal{B}^{\mathcal{G}}_F$ we use the previous transformation to reduce the complement of 3SAT to $\mathcal{B}^{\mathcal{G}}_F$; a 3CNF formula is a $\forall^*\exists^*$QBF with no universal variables. If I_ϕ is the instance obtained from the transformation of ϕ, then $\{t_\exists\}$ is a globally optimal repair of I_ϕ if and only if $\phi \notin 3SAT$. $\square$

Below, we propose two alternative families of preferred repairs with better computational properties than $\mathcal{G}\mathit{Rep}$. One is obtained by relaxing the optimality conditions: it selects a superset of globally optimal repairs. The other is obtained by considering the *"most grounded"* repairs and this family selects a subset of globally optimal repairs. In the following we also state the lower bound on the computational complexity of consistent query answers for general families of preferred repairs.

5.3.2 Pareto optimal repairs

We obtain the first alternative family of preferred repairs by using a notion of optimality that requires a stronger support from the priority to remove a repair from consideration.

Definition 5.15 (Pareto optimal repairs $\mathcal{P}\mathit{Rep}$) A repair I' is *Pareto optimal* if no nonempty subset X of tuples from I' can be replaced with a nonempty set Y of tuples from $I \setminus I'$ such that

$$\forall x \in X. \forall y \in Y. y \succ x$$

and the resulting set of tuples is consistent. $\mathcal{P}\mathit{Rep}(I, F, \succ)$ is the set of all Pareto optimal repairs of I.

Pareto optimality is weaker than global optimality: in Example 1.4 both repairs I_1 and I_3 are Pareto optimal, while only I_1 is globally optimal.

Proposition 5.16 *$\mathcal{P}Rep$ satisfies $\mathcal{P}1$-$\mathcal{P}4$. Also, $\mathcal{G}Rep \sqsubseteq \mathcal{P}Rep$.*

Theorem 5.17 *For any family $\mathcal{X}Rep$ of Pareto optimal repairs satisfying $\mathcal{P}1$ and $\mathcal{P}2$ deciding $\mathcal{X}$-consistent query answering is coNP-hard.*

Proof We show the hardness by reducing the complement of SAT to $\mathcal{D}^{\mathcal{X}}_{F,Q}$.

Take then any CNF formula $\varphi = c_1 \wedge \ldots \wedge c_k$ over variables $x_1, \ldots, x_n$ and let $c_j = l_{j,1} \vee \ldots \vee l_{j,m_j}$. We assume that there are no repetitions of literals in a clause (i.e., $l_{j,k_1} \neq l_{j,k_2}$). We construct a relation instance I_φ over the schema $R(A_1, B_1, A_2, B_2)$ in the presence of two functional dependencies $F = \{A_1 \rightarrow B_1, A_2 \rightarrow B_2\}$. The instance I_φ consists of the following tuples:

- $w_i = R(i, 1, i, 1)$ for to the positive valuation of x_i (for every $i = 1, \ldots, n$),
- $\bar{w}_i = R(i, -1, -i, 1)$ for to the negative valuation of x_i (for every $i = 1, \ldots, n$),
- $v_i^j = R(n + j, 1, -i, 0)$ for the literal x_i in the clause c_j,
- $\bar{v}_i^j = R(n + j, 1, i, 0)$ for the literal $\neg x_i$ in the clause c_j,
- $d_j = R(n + j, 1, 0, 1)$ for the clause c_j (for every $j = 1, \ldots, m$),
- $b = R(0, 0, 0, 0)$ for the formula φ.

The constructed priority $\succ_\varphi$ is the minimal priority on I_φ (w.r.t. F) such that:

$$\begin{array}{lll} \bar{w}_i \succ_\varphi v_i^j, & v_i^j \succ_\varphi d_j, & d_j \succ_\varphi b, \\ w_i \succ_\varphi \bar{v}_i^j, & \bar{v}_i^j \succ_\varphi d_j. & \end{array}$$

The query we consider is $Q = \neg b$.

Figure 5.3 presents prioritized conflict graph obtained from the formula $\varphi = (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_3 \vee \neg x_4 \vee x_5)$.

$$\begin{tikzcd}
\bar{w}_1 \arrow[r, dash] & w_1 \arrow[d] & \bar{w}_2 \arrow[r, dash] \arrow[d] & w_2 & \bar{w}_3 \arrow[r, dash] \arrow[d] \arrow[dr] & w_3 & \bar{w}_4 \arrow[r, dash] & w_4 \arrow[d] & \bar{w}_5 \arrow[r, dash] \arrow[d] & w_5 & \bar{w}_6 \arrow[r, dash] & w_6 \arrow[d] \\
& \bar{v}_1^1 \arrow[drr] & v_2^1 \arrow[dr] & & v_3^1 \arrow[dl] & v_3^2 \arrow[drrr] & & \bar{v}_4^2 \arrow[dr] & v_5^2 \arrow[d] & & & \bar{v}_5^2 \arrow[dlll] \\
& & & d_1 \arrow[drrr] & & & & & d_2 \arrow[dll] & & & \\
& & & & & & b & & & & &
\end{tikzcd}$$

Figure 5.3: The prioritized conflict graph for $\phi = (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_3 \lor \neg x_4 \lor x_5)$.

Now we show that

$$(I_\varphi, \succ_\varphi) \in \mathcal{D}^{\mathcal{X}}_{F,Q} \iff \forall I' \in \mathcal{X}Rep(I_\varphi, F, \succ_\varphi).b \notin I' \iff \varphi \notin SAT.$$

$\Rightarrow$ Suppose there exists a valuation V such that $V \models \varphi$ and consider the following instance

$$\begin{aligned} I' =& \{w_i | V(x_i) = true\} \cup \{\bar{w}_i | V(x_i) = false\} \cup \\ & \{v_i^j | V(x_i) = true\} \cup \{\bar{v}_i^j | V(x_i) = false\} \cup \{b\}. \end{aligned}$$

First, we note that I' is a repair and a Pareto optimal one. Next, we show that $I' \in \mathcal{X}Rep(I_\varphi, F, \succ_\varphi)$. To prove this consider the following priority $\succ' = \succ_\varphi \cup \{(v_i, \bar{v}_i) | V(x_i) = true\} \cup \{(\bar{v}_i, v_i) | V(x_i) = false\}$. We note that I' is the only Pareto optimal repair w.r.t. $\succ'$. By $\mathcal{P}1$ $\mathcal{X}Rep$ is a nonempty family of Pareto optimal repairs and hence $I' \in \mathcal{X}Rep(I_\varphi, F, \succ')$. Since $\succ \subseteq \succ'$ by $\mathcal{P}2$ we get that I' belongs to $\mathcal{X}Rep(I_\varphi, F, \succ)$. Finally, we observe that $b \in I'$ which contradicts $(I_\varphi, \succ_\varphi) \in \mathcal{D}^{\mathcal{X}}_{F,Q}$.

$\Leftarrow$ Suppose there exists a repair $I' \in \mathcal{X}Rep(I_\varphi, F, \succ_\varphi)$ such that $b \in I'$. Obviously, for every $j \in \{1, \ldots, m\}$ the tuple d_j does not belong to I'. Also, for every j at least one tuple neighboring to d_j other than b is present in I' or otherwise I' is not a Pareto optimal repair. Similarly, I' has either w_i or $\bar{w}_i$ for every $i \in \{1, \ldots, n\}$ and hence the following valuation is properly defined:

$$V(x_i) = \begin{cases} true & \text{if } w_i \in I', \\ false & \text{if } \bar{w}_i \in I'. \end{cases}$$

We claim that $V \models \varphi$. Suppose otherwise and take any clause c_j unsatisfied by V. Let $x \neq b$ be the tuple neighboring to d_j that is present in I'. W.l.o.g we can assume that $x = \bar{v}_{j,i_0}$ for some i_0 and then $\neg x_{i_0}$ is a literal of c_j. Also then, w_{i_0} does not belong to I' and so $V(x_{i_0}) = false$. This implies that $V \models \neg x_{i_0}$ and $V \models c_j$; a contradiction.

We finish the proof with the observation that the described reduction requires time polynomial in the size of the instance. □

Corollary 5.18 *$\mathcal{P}$-preferred repair checking is in PTIME and $\mathcal{P}$-preferred consistent query answering is coNP-complete.*

Proof To find if I' is a Pareto optimal repair of I we seek a tuple $z \in I \setminus I'$ whose all neighbors in I' are dominated by z. We claim that such a tuple exists if and only if I' is not Pareto optimal. For the *if* part assume that there exist sets $X \subseteq I'$ and Y such that $\forall x \in X . \forall y \in Y . y \succ x$. Since tuples X are to be replaced by Y, then $Y \subseteq I \setminus I'$. It's sufficient to take any element of Y for z. The *only if* part is trivial. Naturally, the search for z can be performed in time polynomial in the size of the instance.

$\mathcal{D}^{\mathcal{P}}_{F,Q}$ belongs to coNP from the definition of $\mathcal{P}$-preferred consistent query answers and is coNP-complete by Theorem 5.17. □

5.3.3 Common repairs

Definition 5.19 (Common repairs $\mathcal{C}$*Rep*) A repair I' is *common* if and only if $I' \in \mathcal{X}Rep(I, F, \succ)$ for every family $\mathcal{X}Rep$ of Pareto optimal repairs that satisfies $\mathcal{P}1$ and $\mathcal{P}2$. $\mathcal{C}Rep(I, F, \succ)$ is the set of all common repairs of I.

Interestingly, $\mathcal{C}Rep$ has the following procedural characterization. A common repair is constructed by iterative selection of non-dominated tuples, i.e. tuples defined with the *winnow operator* [Cho03]:

$$\omega_{\succ}(I) = \{t \in I | \neg\exists t' \in I. t' \succ t\}.$$

Theorem 5.20 *Fix an instance I and a set of FDs F. For any repair I' and any priority $\succ$ the following statements are equivalent:*

1. *I' is a common repair w.r.t. $\succ$;*
2. *there exists a total priority $\succ'$ extending $\succ$ such that I' is the only Pareto optimal repair w.r.t. $\succ'$;*
3. *I' is a result of Algorithm $\mathcal{PCR}$ (for $\succ$).*

Algorithm $\mathcal{PCR}$: *Prioritized Conflict Resolution*

```
1:  $J \leftarrow I$
2:  $I' \leftarrow \varnothing$
3:  while $\omega_{\succ}(J) \neq \varnothing$ do
4:     choose any $t \in \omega_{\succ}(J)$
5:     $I' \leftarrow I' \cup \{t\}$
6:     $J \leftarrow J \setminus (\{t\} \cup n(t))$ {$n(t)$ is the neighborhood of $t$ in $G(I, F)$.}
7:  return $I'$
```

Proof *(1) ⇒ (2)* Take any $I' \in \mathcal{C}Rep(I, F, \succ)$ and suppose there is no total extension $\succ'$ of $\succ$ such that I' is Pareto optimal w.r.t. $\succ'$. Consider then a family of Pareto optimal repairs which returns all Pareto optimal repairs except I'. This family satisfies $\mathcal{P}1$ and $\mathcal{P}2$. Naturally, I' never belongs to this family of repairs which contradicts that I' is common.

(2) ⇒ (3) Let $|I'| = n$ and assume there is a total priority $\succ' \supseteq \succ$ such that I' is Pareto optimal w.r.t. $\succ'$.

First, we observe that by $\mathcal{G}Rep \sqsubseteq \mathcal{P}Rep$ and $\mathcal{P}4$ for $\mathcal{G}Rep$ the repair I' is also the only globally optimal repair w.r.t. $\succ'$.

Next, we show that I' is a result of Algorithm $\mathcal{PCR}$ for $\succ'$. For that, we construct the following sequence $x_1, \ldots, x_n$ of choices necessary to make in Step 5.20:

$$x_i \text{ is any element of } \omega_{\succ'}\big(I \setminus (\{x_1, \ldots, x_{i-1}\} \cup n(x_1) \cup \ldots \cup n(x_{i-1}))\big) \cap I'.$$

This sequence is well defined. Otherwise, take the minimal i such that the set $\omega_{\succ'}(I \setminus (\{x_1, \ldots, x_{i-1}\} \cup n(x_1) \cup \ldots \cup n(x_{i-1}))) \cap I'$ is empty, and let $X = \{x_1, \ldots, x_{i-1}\}$ and $Y = I \setminus (\{x_1, \ldots, x_{i-1}\} \cup n(x_1) \cup \ldots \cup n(x_{i-1}))$. Then, $\forall x \in X. \exists y \in Y. y \succ' x$, which contradicts global optimality of I'.

Obviously, with the sequence $x_1, \ldots, x_n$ of choices made in Step 5.20 the repair I' is the result of Algorithm $\mathcal{PCR}$ for $\succ'$. We finish the proof by observing that using a more general priority in Step 5.20 does not restrict possible choices, i.e. $\omega_{\succ'}(s) \subseteq \omega_{\succ}(s)$ for any $\succ' \supseteq \succ$. Hence I' can be constructed with Algorithm $\mathcal{PCR}$ for $\succ$ with the same sequence of choices.

(3) ⇒ (2) Assume that I' is a result of Algorithm $\mathcal{PCR}$ and let $x_1^0, \ldots, x_n^0$ be the sequence of consecutive choices made in Step 5.20.

For every x_i^0 let $x_i^1, \ldots, x_i^{m_i}$ be any ordering of $n(x_i^0) \setminus (\{x_1^0, \ldots, x_i^0\} \cup n(x_1^0) \cup \ldots \cup n(x_{i-1}^0))$ that agrees with $\succ$ (any topological sorting). We note that if $x_i^j \in I \setminus I'$, then $j > 0$.

Next, we construct the following binary relation on I:

$$x_{i_1}^{j_1} \succ' x_{i_2}^{j_2} \iff \{x_{i_1}^{j_1}, x_{i_2}^{j_2}\} - \text{conflict} \wedge i_1 < i_2 \vee (i_1 = i_2 \wedge j_1 < j_2).$$

It is easy to see that $\succ'$ is a priority (i.e. is acyclic), is total, and extends $\succ$.

Now, we show that I' is Pareto optimal w.r.t. $\succ'$. Suppose otherwise, i.e. that there is a nonempty set $X \subseteq I'$ and a nonempty set $Y \subseteq I \setminus I'$ such that $(I' \setminus X) \cup Y$ is consistent and $\forall x \in X. \forall y \in Y. y \succ' x$. We select the element $x_{i_1}^0 \in X$ with the minimum i_1 index and any element $x_i^j \in Y$. We note that $j > 0$ and $x_i^j \in n(x_{i_0}^0)$ because $x_{i_0}^0$ and x_i^j create a conflict and $x_i^j \in I \setminus I'$. Also, because $(I' \setminus X) \cup Y$ is consistent, x_i^j conflicts with no $x_{i'}^0$ for any $i' < i_0$. Hence, from the construction of $\succ'$ we have that $x_{i_0}^0 \succ' x_i^j$; a contradiction.

(2) ⇒ (1) Assume that I' is Pareto optimal w.r.t. a total priority $\succ' \supseteq \succ$. We note that $\mathcal{P}4$ for $\mathcal{P}Rep$ implies that I' is the only Pareto optimal repair w.r.t. $\succ'$. Thus, every family of Pareto optimal repair satisfying $\mathcal{P}1$ and $\mathcal{P}2$ must contain I'. □

Proposition 5.21 *$\mathcal{C}Rep$ satisfies $\mathcal{P}1$-$\mathcal{P}4$ and $\mathcal{C}Rep \sqsubseteq \mathcal{G}Rep$.*

The introduced families of preferred repairs create a hierarchy:

$$\mathcal{C}\mathit{Rep} \sqsubseteq \mathcal{G}\mathit{Rep} \sqsubseteq \mathcal{P}\mathit{Rep}.$$

Recall from the previous section that $\mathcal{P}\mathit{Rep} \neq \mathcal{G}\mathit{Rep}$. The following example shows also that $\mathcal{C}\mathit{Rep} \neq \mathcal{G}\mathit{Rep}$. Thus, the hierarchy is proper.

Example 5.22 Consider a relational instance $I = \{t_a, t_b, t_c, t_d\}$ whose prioritized conflict graph is presented in Figure 5.4 (for brevity we omit the exact values of the tuples and the constraints). The instance I has 3 repairs: $I_1 = \{t_a\}$, $I_2 = \{t_b\}$, and $I_3 = \{t_c, t_d\}$. All

$$\begin{tikzcd}
t_a \arrow[r, dash] \arrow[d] \arrow[dr, dash] & t_b \arrow[d] \arrow[dl, dash] \\
t_c & t_d
\end{tikzcd}$$

Figure 5.4: The prioritized conflict graph $G(I, F, \succ)$.

repairs are globally optimal, but only I_1 and I_2 are common.

We observe, however, that under certain conditions this hierarchy collapses.

Proposition 5.23 *$\mathcal{P}\mathit{Rep}$, $\mathcal{G}\mathit{Rep}$, and $\mathcal{C}\mathit{Rep}$ coincide under one of the following conditions:*

1. *the set of constraints F consists of one key dependency only;*
2. *the priority $\succ$ can be extended to acyclic orientations only.*

Moreover, $\mathcal{G}\mathit{Rep}$ and $\mathcal{C}\mathit{Rep}$ coincide if

(3) the set of constraints F consists of one functional dependency only.

Proof In the cases *(1)* and *(2)* it's enough to show $\mathcal{P}\mathit{Rep} \sqsubseteq \mathcal{C}\mathit{Rep}$.

(1) We use the fact that in the presence of one key dependency only the conflict graph is a union of pairwise disjoint cliques and every repair consists of one element selected from each clique.

We fix an instance I, a key dependency F, and a priority $\succ$. Let $c_1, \ldots, c_n$ be the cliques of $G(I, F)$. Take any $I' \in \mathcal{P}\mathit{Rep}(I, F, \succ)$ and let $t_1, \ldots, t_n$ be the elements of I' such that

$t_i \in c_i$. We note that since I' is Pareto optimal, then for every i any $y \in c_i \setminus \{t_i\}$ $y \not\succ t_i$ and consequently $t_i \in \omega_\succ(c_i)$. Hence, $t_1, \ldots, t_n$ is a proper sequence of choices in Step 5.20 of Algorithm $\mathcal{PCR}$. To finish the proof we note that $n(t_i) \cup \{t_i\} = c_i$ which implies that Algorithm $\mathcal{PCR}$ halts after consecutively choosing the elements $t_1, \ldots, t_n$.

(2) We take any $I' \in \mathcal{P}Rep(I, F, \succ)$ and construct a total extension $\succ'$ of $\succ$ by prioritizing conflicts unprioritized by $\succ$ in favor of I', i.e. $\succ'$ is any total priority such that for any $x \in I'$ and any y conflicting with x if $y \not\succ x$ then $x \succ' y$. Since $\succ$ can be extended to acyclic orientations only, $\succ'$ is acyclic.

Clearly, I' is a Pareto optimal repair w.r.t $\succ'$ and a unique one (by $\mathcal{P}4$). Therefore $I' \in \mathcal{C}Rep(I, F, \succ')$ and by $\mathcal{P}2$ we get $I' \in \mathcal{C}Rep(I, F, \succ)$.

(3) We show that $\mathcal{G}Rep \sqsubseteq \mathcal{C}Rep$ if we assume a single functional dependency $F = \{X \rightarrow Y\}$ and we use the notions X*-cluster* and (X, Y)*-cluster* [ABC+03b] for an FD $R : X \rightarrow Y$. An (X, Y)-cluster is a maximal set of tuples of R in I that have the same attribute value in X and Y. An X-cluster is the union of all (X, Y)-clusters with the same attribute value in X. Clearly, every repair contains exactly one (X, Y)-cluster from each X-cluster. We also note that conflicts are present only inside an X-cluster and two tuples from X-cluster form a conflict if and only if they belong to two different (X, Y)-clusters.

Now, let the instance I be the sum of the X-clusters $C_1, \ldots, C_n$. Take any globally optimal repair and let it be the sum of the (X, Y)-clusters $D_1, \ldots, D_n$ ($d_i \subseteq c_i$ for every i). By global optimality of I' we have that for every i

$$\exists t_i \in D_i . \forall y \in C_i \setminus D_i . y \not\succ t_i.$$

Therefore, Algorithm $\mathcal{PCR}$ can perform the first n iterations by choosing $t_1, \ldots, t_n$. Because $n(T_i) = C_i \setminus D_i$ and elements of D_i conflict only with elements of $C_i \setminus D_i$, the remaining choices can consist of any ordering of $(D_1 \setminus \{t_1\}) \cup \ldots \cup (D_n \setminus \{t_n\})$. Hence, I' is a result of Algorithm $\mathcal{PCR}$. □

We note that the conditions are necessary but not sufficient; for instance, the priority in Figure 5.1 can be extended to a cyclic orientation, yet $\mathcal{G}Rep$ and $\mathcal{C}Rep$ coincide.

Corollary 5.24 *$\mathcal{C}$-preferred repair checking is in PTIME and $\mathcal{C}$-preferred consistent query* *answering is coNP-complete.*

Proof To check if a repair I' is common we use Algorithm $\mathcal{PCR}$ to simulate the construction of I' by restricting the choice in Step 5.20 to tuples $\omega_{\succ}(J) \cap I'$. The repair I' is common if and only if such a simulation can be performed successfully (i.e. it produces I'). Naturally, $\mathcal{D}^{\mathcal{C}}_{F,Q}$ belongs to coNP and its coNP-completeness follows from Theorem 5.17. $\square$

5.3.4 Positive case

The intractability proofs use at least 2 FDs. Next, we investigate the case when only one FD is present. Because for one FD the introduced families collapse (Proposition 5.23) we do not need to differentiate between them.

Theorem 5.25 *If the set of integrity constraints contains at most one functional dependency per relation name and no other constraints, then computing preferred consistent answers to quantifier-free queries is in PTIME for $\mathcal{P}$Rep, $\mathcal{G}$Rep, and $\mathcal{C}$Rep.*

Proof First, we observe that if only functional dependencies are considered, tuples can create conflict only with tuples from the same relation and therefore we can limit our consideration to schema consisting on one relation name only.

We assume that the set of integrity constraints is $F = \{R : X \to Y\}$ and we use the notions X-*cluster* and (X,Y)-*cluster* (see the proof of Proposition 5.23).

Now, we fix an instance I and a priority $\succ$. For every tuple $t \in I$, by C_t we denote the X-cluster t belongs to and by D_t we denote its (X,Y)-cluster.

We adopt the algorithm from [CM05]. We assume that the query is in CNF, i.e. $\Phi = \Phi_1 \wedge \ldots \wedge \Phi_n$. *True* is not a preferred consistent query answer to Φ in I if and only if there exists a preferred repair I' and there exists i such that $I' \not\models \Phi_i$. The algorithm attempts to find if such a repair exists for every i. Fix i and consider

$$\neg\Phi_i = t_1 \wedge \ldots \wedge t_k \wedge \neg t_{k+1} \wedge \ldots \wedge \neg t_m.$$

To find if a preferred repair satisfying $\neg\Phi_i$ exists we use one of the two following polynomial tests depending on the family of preferred repairs we use. For simplicity, we assume that the tuples $t_1, \ldots, t_k$ and the tuples $t_{k+1}, \ldots, t_n$ belong to I; otherwise there is no repair satisfying $\neg\Phi_i$ or we can remove the negative literal from $\neg\Phi_i$ respectively (because repairs are subsets of I). Recall that globally optimal and common repairs coincide in the presence of one FD.

Lemma 5.26 *A globally optimal (common) repair I' satisfying $\neg\Phi_i$ exists if and only if the following conditions are satisfied:*

1. *$\{t_1, \ldots, t_k\}$ is conflict-free;*
2. *$\{D_{t_1}, \ldots, D_{t_k}\} \cap \{D_{t_{k+1}}, \ldots, D_{t_m}\} = \varnothing$;*
3. *$D_{t_j} \cap \omega_{\succ}(C_{t_j}) \neq \varnothing$ for every $j \in \{1, \ldots, k\}$.*
4. *$\omega_{\succ}(C_{t_j}) \setminus (D_{t_{k+1}} \cup \ldots \cup D_{t_n}) \neq \varnothing$ for every $j \in \{k+1, \ldots, n\}$.*

Lemma 5.27 *A Pareto optimal repair I' satisfying $\neg\Phi_i$ exists if and only if the following conditions are satisfied:*

1. *$\{t_1, \ldots, t_k\}$ is conflict-free;*
2. *$\{D_{t_1}, \ldots, D_{t_k}\} \cap \{D_{t_{k+1}}, \ldots, D_{t_m}\} = \varnothing$;*
3. *for every $j \in \{1, \ldots, k\}$, for every tuple $t \in C_{t_j} \setminus D_{t_j}$ there exists $t' \in D_{t_j}$ such that $t \not\succ t'$.*
4. *for every $j \in \{k+1, \ldots, n\}$ there exists an (X, Y)-cluster D of C_{t_j} different from $D_{t_{k+1}}, \ldots, D_{t_n}$ such that for every $t \in D_{t_{k+1}} \cup \ldots \cup D_{t_n}$, there exists $t' \in D$ such that $t \not\succ t'$.*

Full proofs of these Lemmas are in the appendix. Here, we just observe that by 1 and 3 we can construct a preferred repair containing $t_1, \ldots, t_k$ and by 2 and 4 the preferred repair does not contain $t_{k+1}, \ldots, t_m$. $\square$

5.4 Related work

We limit our discussion to the work on using priorities to maintain consistency and facilitate resolution of conflicts.

The first article to notice the importance of priorities in information systems is [FUV83]. There, the problem of conflicting updates in (propositional) databases is solved in a manner similar to $\mathcal{C}Rep$. The considered priorities are transitive, which in our framework is too restrictive. Also, in our framework this restriction does not bring any computational benefits (the reductions can be modified to use only transitive priorities). [Bre89] is another example of $\mathcal{C}Rep$-like prioritized conflict resolution of first-order theories. The basic framework is defined for priorities which are weak orders. A partial order is handled by considering every extension to weak order. This approach also assumes the transitivity of the priority.

In the context of logic programs, priorities among rules can be used to handle inconsistent logic programs (where rules imply contradictory facts). More preferred rules are satisfied, possibly at the cost of violating less important ones. In a manner analogous to Proposition 5.7, [VNV02] lifts a total order on rules to a preference on (extended) answers sets. When computing answers only maximally preferred answers sets are considered.

A simpler approach to the problem of inconsistent logic programs is presented in [Gro97]. There, conflicting facts are removed from the model unless the priority specifies how to resolve the conflict. Because only programs without disjunction are considered, this approach always returns exactly one model of the input program. Constructing preferred repairs in a corresponding fashion (by removing all conflicts unless the priority indicates a resolution) would similarly return exactly one database instance (fulfillment of $\mathcal{P}1$ and $\mathcal{P}4$). However, if the priority is not total, the returned instance is not a repair and therefore $\mathcal{P}5$ is not satisfied. Such an approach leads to a loss of (disjunctive) information and does not satisfy $\mathcal{P}2$ and $\mathcal{P}3$.

[FGZ04] proposes a framework of *conditioned active integrity constraints*, which allows the user to specify the way some of the conflicts created with a constraint can be resolved. This framework satisfies properties $\mathcal{P}1$ and $\mathcal{P}2$ and doesn't satisfy $\mathcal{P}3$ and $\mathcal{P}4$. [FGZ04] also describes how to translate conditioned active integrity constraints into a prioritized logic program [SI00], whose preferred models correspond to maximally preferred repairs.

[MAA04] uses ranking functions on tuples to resolve conflicts by taking only the tuple with highest rank and removing others. This approach constructs a unique repair under the assumption that no two different tuples are of equal rank (satisfaction of $\mathcal{P}4$). If this assumption is not satisfied and the tuples contain numeric values, a new value, called the fusion, can be calculated from the conflicting tuples (then, however, the constructed instance is not necessarily a repair in the sense of Definition 2.4 which means a possible loss of information).

A different approach based on ranking is studied in [GSTZ04]. The authors consider polynomial functions that are used to rank repairs. When computing preferred consistent query answers, only repairs with the highest rank are considered. The properties $\mathcal{P}2$ and $\mathcal{P}5$ are trivially satisfied, but because this form of preference information does not have natural notions of extensions and maximality, it is hard to discuss postulates $\mathcal{P}3$ and $\mathcal{P}4$. Also, the preference among repairs in this method is not based on the way in which the conflicts are resolved.

An approach where the user has a certain degree of control over the way the conflicts are resolved is presented in [GL04]. Using repair constraints the user can restrict considered repairs to those where tuples from one relation have been removed only if similar tuples have been removed from some other relation. This approach satisfies $\mathcal{P}3$ but not $\mathcal{P}1$. A method of weakening the repair constraints is proposed to get $\mathcal{P}1$, however this comes at the price of losing $\mathcal{P}3$.

In [AFM06], Andritsos et al. extend the framework of consistent query answers with techniques of probabilistic databases. Essentially, only one key dependency per relation is considered and user preference is expressed by assigning a probability value to each of mutually conflicting tuples. The probability values must sum to 1 over every clique in the conflict graphs. This framework generalizes the standard framework of consistent query answers: the repairs correspond to possible worlds and have an associated probability. We also note that no repairs are removed from consideration (unless the probability of the world is 0). The query is evaluated over all repairs and the probability assigned to an answer is the sum of probabilities of worlds in which the answer is present. Although the considered databases are repairs, the use of the associated probability values makes it difficult to compare this framework with ours.

Chapter 6

XML databases

6.1 Data model

We adapt a model of XML documents and DTDs similar to those commonly used in the literature [BPV04, BFG05, KSS03, Nev02].

Ordered labeled trees We view XML documents as labeled ordered trees with text values. For simplicity we ignore attributes: they can be easily simulated using text values. By Σ we denote a fixed (and finite) set of node labels and we distinguish a label $\texttt{PCDATA} \in \Sigma$ to identify *text nodes*. A text node has no children and is additionally labeled with an element from an infinite domain Γ of text constants. For clarity of presentation, we use capital letters $\texttt{A}, \texttt{B}, \texttt{C}, \ldots$ for elements from Σ and capital letters $X, Y, Z \ldots$ for variables ranging over Σ. We assign a unique identifier to every node. Figure 6.1 contains an example of a tree.

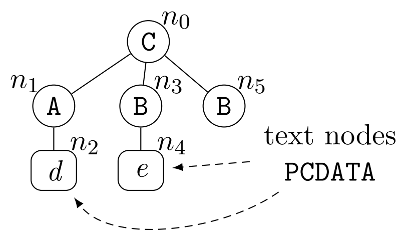


Figure 6.1: Running example

We assume that the data structure used to store a document allows for any given node to get its label, its parent, its first child, and its immediate following sibling in time $O(1)$.

For the purpose of presentation, we represent trees as terms over the signature $\Sigma \setminus \{\texttt{PCDATA}\}$ with constants from Γ. For instance, the tree T_1 from Figure 6.1 is represented as $\texttt{C}(\texttt{A}(\texttt{d}), \texttt{B}(\texttt{e}), \texttt{B})$. Note that a text constant $t \in \Gamma$, viewed as a term, represents a tree node with no children, that is labeled with PCDATA, and whose additional text label is t.

DTDs For simplicity our view of DTDs omits the specification of the root label. A *DTD* is a function D that maps labels from $\Sigma \setminus \{\texttt{PCDATA}\}$ to regular expressions over Σ. We use *regular expressions* given by the grammar

$$E ::\equiv \epsilon \,|\, X \,|\, E + E \,|\, E \cdot E \,|\, E^*,$$

where X ranges over Σ, ϵ denotes the empty string, and $E + E$, $E \cdot E$, E^* denote the union, concatenation and the Kleene closure, respectively. $L(E)$ is the set of all strings over Σ satisfying E. $|E|$ stands for the size (length) of E. The size of D, denoted $|D|$, is the sum of the lengths of the regular expressions occurring in D.

A tree $T = X(T_1, \dots, T_n)$ is *valid* w.r.t. a DTD D if: (1) T_i is valid w.r.t. D for every i and, (2) if $X_1, \dots, X_n$ are labels of root nodes of $T_1, \dots, T_n$ respectively and $E = D(X)$, then $X_1 \cdots X_n \in L(E)$.

Example 6.1 Consider the following DTD D_1:

$$D_1(\texttt{C}) = (\texttt{A} \cdot \texttt{B})^*, \quad D_1(\texttt{A}) = \texttt{PCDATA} + \epsilon, \quad D_1(\texttt{B}) = \epsilon.$$

The tree $T_1 = \texttt{C}(\texttt{A}(\texttt{d}), \texttt{B}(\texttt{e}), \texttt{B})$ is not valid w.r.t. D_1 but the tree $\texttt{C}(\texttt{A}(\texttt{d}), \texttt{B})$ is.

To capture the sets of strings satisfying regular expressions we use the standard notion of *non-deterministic finite automaton* (NFA) $M = \langle \Sigma, S, q_0, \Delta, F \rangle$, where S is a finite set of *states*, $q_0 \in S$ is a distinguished *starting state*, $F \subseteq S$ is the set of *final states*, and $\Delta \subseteq S \times \Sigma \times S$ is the *transition relation*. We note that we consider only ϵ-free NFAs and recall the classic result that for every regular expression there exists an equivalent ϵ-free NFA whose set of states is of size linear in the size of the expression [HMU01].

6.1.1 Tree edit distance and repairs

A *location* is a sequence of natural numbers defined as follows: ε is the location of the root node, and $v \cdot i$ is the location of i-th child of the node at location v. This notion allows us to identify nodes without fixing a tree.

We consider the three standard *tree operations* [AMR+98, BPV04, BFG05, CRGMW96, CAM02]:

1. *Deleting* a subtree rooted at a specified location.
2. *Inserting* a subtree at a specified location.
3. *Modifying* the label at a specified location.

With each operation we associate its cost. For deleting (inserting) it is the size of the deleted (resp. inserted) subtree. The cost of label modification is 1.

The following example shows that the order of applying operations is important, and therefore we consider sequences (rather than sets) of operations when transforming documents.

Example 6.2 Recall the tree $T_1 = \mathtt{C}(\mathtt{A}(\mathtt{d}), \mathtt{B}(\mathtt{e}), \mathtt{B})$ from Figure 6.1. If we first insert $\mathtt{D}$ as a second child of the root and then remove the first child of the root we obtain $\mathtt{C}(\mathtt{D}, \mathtt{B}(\mathtt{e}), \mathtt{B})$. If, however, we first remove the first child of the root and then add $\mathtt{D}$ as a second child of the root we get $\mathtt{C}(\mathtt{B}(\mathtt{e}), \mathtt{D}, \mathtt{B})$.

The *cost* of a sequence of operations is the sum of costs of its elements. Two sequences of operations are *equivalent* on a tree T if their application to T yields the same tree. We observe that some sequences may perform redundant operations, for instance an insertion and subsequent removal of a subtree. Because we focus on finding cheapest sequences of operations, we restrict our considerations to *redundancy-free* sequences (those for which there is no equivalent but cheaper sequence). We also observe that some sequences of operations may be unnecessarily long, for instance a subtree can inserted with a sequences of operations inserting single nodes rather than one operation inserting the whole subtree at one time. Both ways are equivalent and their cost is the same, therefore we further restrict our considerations to *concise* sequences of operations (those for which there is no equivalent but shorter sequence).

Definition 6.3 (Edit distance) Given two trees T and T', the *edit distance* $dist(T, T')$ between T and T' is the minimum cost of transforming T into T'.

Note that the distance between two documents (even if we consider only deletions and insertions) is a metric i.e., it is positively defined, symmetric, and satisfies the triangle inequality.

For a DTD D and a (possibly invalid) tree T, a sequence of operations is a sequence *repairing* T w.r.t. D if the document resulting from applying the sequence to T is valid w.r.t. D. We are interested in the optimal repairing sequences of T.

Definition 6.4 (Distance to a DTD) Given a document T and a DTD D, the *distance* $dist(T, D)$ of T to D is the minimum cost of transforming T into a document valid w.r.t. D.

Repairs

We use the notions of edit distance to capture the minimality of change required to repair a document.

Definition 6.5 (Repair) Given a document T and a DTD D, a document T' is a *repair* of T w.r.t. D if T' is valid w.r.t. D and $dist(T, T') = dist(T, D)$.

Note that, if repairing a document involves inserting a text node, the corresponding text label can have infinitely many values, and thus in general there can be infinitely many repairs. However, as shown in the following example, even if the operations are restricted to deletions there can be an exponential number of non-isomorphic repairs of a given document.

Example 6.6 Consider the following DTD D_2:

$$D_2(\mathtt{A}) = (\mathtt{B} \cdot (\mathtt{T} + \mathtt{F}))^*, \qquad D_2(\mathtt{T}) = \epsilon,$$

$$D_2(\mathtt{B}) = \mathtt{PCDATA}, \qquad D_2(\mathtt{F}) = \epsilon.$$

The document $\mathtt{A}(\mathtt{B}(\mathtt{1}), \mathtt{T}, \mathtt{F}, \ldots, \mathtt{B}(\mathtt{n}), \mathtt{T}, \mathtt{F})$ consists of $4n+1$ elements and has 2^n repairs w.r.t. D_2. For instance, an example of a repair for the document $T_2 = \mathtt{A}(\mathtt{B}(\mathtt{1}), \mathtt{T}, \mathtt{F}, \mathtt{B}(\mathtt{2}), \mathtt{T}, \mathtt{F}, \mathtt{B}(\mathtt{3}), \mathtt{T}, \mathtt{F})$ is the document $\mathtt{A}(\mathtt{B}(\mathtt{1}), \mathtt{T}, \mathtt{B}(\mathtt{2}), \mathtt{F}, \mathtt{B}(\mathtt{3}), \mathtt{T})$.

6.2 Trace graph

In this section we present a construction that allows to capture all repairs of an XML document. For clarity of presentation, we limit operations to insertion and deletion. Later on we show how to extend our approach to handle label modification as well.

The main element of this construction is a *trace graph* which is built for every node of the tree. A trace graph captures all optimal ways to repair the document at a given node.

6.2.1 Restoration graph

We start the construction of a trace graph by building first a *restoration graph* which captures all (not necessarily optimal) ways to repair the document at a given node.

Suppose $T = X(T_1, \ldots, T_n)$ is a tree and $X_1, \ldots, X_n$ the sequence of the root labels of $T_1, \ldots, T_n$ respectively. The DTD we work with is D and $E = D(X)$. Let $M_E = \langle \Sigma, S, q_0, \Delta, F \rangle$ be the NDFA recognizing $L(E)$.

The *restoration graph* for the root of T, denoted U_T, contains vertices of the form q^i for $q \in S$ and $i \in \{0, \ldots, n\}$. The vertex q^i is referred as the *state q in the i-th column of* U_T. q^i corresponds to the state q being reached by M_E after reading the elements $X_1, \ldots, X_i$ with possibly some elements deleted and some elements inserted. The edges of U_T are:

- $p^{i-1} \xrightarrow{\mathit{Read}} q^i$ exists only if $\Delta(p, X_i, q)$ (we read X_i from the input and change the state of M_E accordingly);
- $q^{i-1} \xrightarrow{\mathit{Del}} q^i$ for any state $q \in S$ and any $i \in \{1, \ldots, n\}$ (we remove X_i from the input but we don't change the state of M_E);
- $p^i \xrightarrow{\mathit{Ins}\ Y} q^i$ exists only if $\Delta(p, Y, q)$ (we don't remove any symbol from the input but we change the state of M_E as if the symbol Y was read).

A *repairing path* in U_T is a path from q_0^0 to any accepting state in the last column of U_T.

Example 6.7 Recall the document $T_1 = \mathtt{C}(\mathtt{A}(\mathtt{d}), \mathtt{B}(\mathtt{e}), \mathtt{B})$ and the DTD D_1 from Example 6.1. The automaton $M_{(\mathtt{A}\cdot\mathtt{B})^*}$ consists of two states q_0 and q_1; q_0 is both the starting and the only accepting state; $\Delta = \{(q_0, \mathtt{A}, q_1), (q_1, \mathtt{B}, q_0)\}$. The restoration graph U_T is presented in Figure 6.2.

$$\begin{tikzcd}
q_0^0 \arrow[r, "\mathit{Del}"] \arrow[d, bend right, "\mathit{Ins}\ \mathtt{A}"'] \arrow[dr, "\mathit{Read}"] & q_0^1 \arrow[r, "\mathit{Del}"] \arrow[d, bend right, "\mathit{Ins}\ \mathtt{A}"'] & q_0^2 \arrow[r, "\mathit{Del}"] \arrow[d, bend right, "\mathit{Ins}\ \mathtt{A}"'] & q_0^3 \arrow[d, bend right, "\mathit{Ins}\ \mathtt{A}"'] \\
q_1^0 \arrow[r, "\mathit{Del}"] \arrow[u, bend right, "\mathit{Ins}\ \mathtt{B}"'] & q_1^1 \arrow[r, "\mathit{Del}"] \arrow[u, bend right, "\mathit{Ins}\ \mathtt{B}"'] \arrow[ur, "\mathit{Read}"] & q_1^2 \arrow[r, "\mathit{Del}"] \arrow[u, bend right, "\mathit{Ins}\ \mathtt{B}"'] \arrow[ur, "\mathit{Read}"] & q_1^3 \arrow[u, bend right, "\mathit{Ins}\ \mathtt{B}"']
\end{tikzcd}$$

Figure 6.2: Construction of the restoration graph

We note that the restoration graph is constructed for every node of the tree T. Now, we show how to obtain a sequence of editing operations equivalent to a selection of a path in every restoration graph of T. We translate recursively the path in U_T as follows:

- $q^{i-1} \xrightarrow{\mathit{Del}} q^i$ is mapped to a deletion of T_i,
- $p^i \xrightarrow{\mathit{Ins}\ Y} q^i$ is mapped to an insertion of a valid subtree with root label Y,
- $p^{i-1} \xrightarrow{\mathit{Read}} q^i$ is mapped to a repairing sequence for T_i (obtained by induction from U_{T_i}).

Lemma 6.8 *Given a tree T and a DTD D, for every repairing sequence of editing operations there exists an equivalent selection of repairing path in every restoration graph of T, and vice versa.*

Proof We first observe that for every non-redundant and concise sequence of editing operations on T there exists an equivalent *canonical* sequence of editing operations, i.e. a sequence of operation whose locations of subsequent editing operations follow the natural order of the document (left-to-right prefix traversal). Because we consider only non-redundant and concise sequences, every sequence can be transformed to an equivalent canonical one by sorting (with possible updates of the locations caused by changes of the relative positions of the editing operations).

A canonical repairing sequence of operations can be partitioned into segments operating on the subsequent subtrees of T: either repairing, inserting, or deleting a subtree. Naturally, those segments correspond to equivalent edges in the restoration graph. Because we use non-deterministic automata, a segment may correspond to more than one edge in U_T, however, all corresponding edges represent the same operation. Finally, because we deal with a repairing sequence of operations we can construct from the corresponding edges an equivalent repairing path in U_T.

For a repairing path in U_T we construct a sequence of editing operations in the fashion presented above. Naturally, we obtain a repairing sequence. □

6.2.2 Compact representation of repairs

Now, to capture optimal repairing sequences of T we assign to the edges of U_T the (minimum) costs of the corresponding operations.

Definition 6.9 (Trace graph) The *trace graph* for T, denoted U_T^*, is the subgraph of U_T consisting of only the optimal repairing paths. The exact cost assigned to edges of U_T is:

- $q^{i-1} \xrightarrow{Del} q^i$ the cost is $|T_i|$,
- $q^i \xrightarrow{Ins\,Y} p^i$ the cost is equal to the minimal size of a valid subtree with root label Y (this cost can be computed with a simple algorithm omitted here),
- $q^{i-1} \xrightarrow{Read} p^i$ the cost is the cost of the shortest repairing path in $U_{T_i}^*$ (obtained by recursion).

Theorem 6.10 *The distance between the tree T and the DTD D is the cost of an optimal repairing path in U_T^*.*

Proof By Lemma 6.8 for every repairing sequence of editing operations there is an equivalent repairing path in the trace graph. We note that because we consider only non-redundant and concise sequences of editing operations, the cost of the sequence of operations and the corresponding path in the trace graph are the same. Also an optimal repairing sequence of editing operation inserts only a minimal valid subtrees, or otherwise it is not optimal.

Therefore, the cost of optimal repairing sequence is equal to the cost of the shortest path in the trace graph. □

Example 6.11 (cont. Example 6.7) Repairing the second child of T_1 requires removing the text node `d` and hence the cost assigned to $q_1^1 \xrightarrow{Read} q_0^2$ is 1. For the DTD D_1 all insertion costs are 1. The trace graph U_T^* is presented in Figure 6.3. The repairing paths in the trace

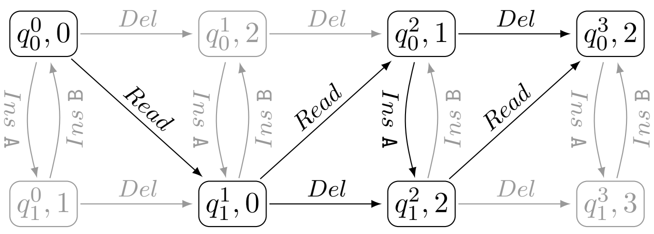


Figure 6.3: Construction of the trace graph

graph capture 3 different repairs:

1. $q_0^0 \xrightarrow{Read} q_1^1 \xrightarrow{Read} q_0^2 \xrightarrow{Ins\ \mathtt{A}} q_1^2 \xrightarrow{Read} q_0^3$ yields `C(A(d), B, A, B)` by repairing the second child (which involves deleting the text node `e`), and inserting `A`;
2. $q_0^0 \xrightarrow{Read} q_1^1 \xrightarrow{Del} q_1^2 \xrightarrow{Read} q_0^3$ yields `C(A(d), B)` by deleting the second child;
3. $q_0^0 \xrightarrow{Read} q_1^1 \xrightarrow{Read} q_0^2 \xrightarrow{Del} q_0^3$ yields `C(A(d), B)` by repairing the second child and deleting the third.

We note that although isomorphic, the repairs (2) and (3) are not the same because the nodes labeled with `B` are obtained from different nodes in the original tree (resp. n_5 and n_3).

Repairs and paths in the trace graph

Note that if U_T has cycles, only *Ins* edges can be involved in them. Since the costs of inserting operations are positive, U_T^* is a directed acyclic graph.

Suppose now that we have constructed a trace graph in every node of T. Every repair can be characterized by selecting a repairing path in each of the trace graphs. Similarly a choice of paths in each of the trace graphs yields a repair. It is important to note that a choice of a path on the top-level trace graph of T may correspond to more than one repair (this happens when some subtree has more than one repair). Trace graphs can also be used for interactive document repair.

Complexity analysis We note that for any vertex from the restoration graph U_T the incoming edges come from the same or the preceding column. Therefore, when computing the minimum cost of a path to a given vertex we need to consider at most $|\Sigma| \times |S| + |S| + 1$ values (up to $|\Sigma| \times |S|$ *Ins* edges, up to $|S|$ *Read* edges, and one *Del* edge). We assume that $|S|$ is bounded by the size of D and that Σ is fixed.

Theorem 6.12 *For a given tree T and a DTD D all trace graphs of T can be constructed in $O(|D|^2 \times |T|)$ time.*

6.2.3 Handling label modification

In order to extend the trace graph to handle label modification we add the following edges:

- $q^i \xrightarrow{Mod\ Y} p^{i+1}$ which exists only if $\Delta(q, Y, p)$ and $Y \neq X_i$ (we remove X_i from the input, but we change the state of M_E as if Y was read);

This edges corresponds to modification of the root label of T_i to Y and then recursively repairing T_i' (which is T_i with the root label changed to Y). The cost assigned to this edge is equal $1 + dist(T_i', D)$. We obtain the value $dist(T_i', D)$ similarly to the way we obtained $dist(T_i, D)$: by first constructing the trace graph $U^*_{T_i'}$.

Handling label modification requires computing $|\Sigma|$ values rather than 1, but since Σ is assumed to be fixed, the asymptotic time and space complexity remains the same. Note, however, that in this way a significant constant factor in the real running time of the algorithm is introduced.

6.3 Validity-sensitive querying

We consider the class $\mathcal{X}$Core of *Core XPath* queries [Mar04] which is known to capture XPath 1.0 [W3C99] restricted to its logical core (all axes but no attributes and no functions). This class of queries is defined with the following grammar:

$$\begin{aligned}
Q &::\equiv p_1 \cup \ldots \cup p_n,\\
p &::\equiv s_1/s_2/\ldots/s_m,\\
s &::\equiv a{::}n \,|\, a{::}n[f],\\
a &::\equiv \epsilon \,|\, \Downarrow \,|\, \Uparrow \,|\, \Leftarrow \,|\, \Rightarrow \,|\, a^*,\\
n &::\equiv * \,|\, X \,|\, text(),\\
f &::\equiv q \,|\, q = str \,|\, q_1 = q_2 \,|\, \neg f \,|\, f_1 \wedge f_2 \,|\, f_1 \vee f_2.
\end{aligned}$$

A query is a union of path and every path is a sequence of steps. Every step consists of axis specification, a node test and an optional filter expression. A node test can be either the wildcard test $*$, a tag name from Σ, or text node test. A filter expression is a boolean expression composed of other queries, textual value tests, and join conditions. We observe that the used set of axes allows to express all standard XPath axes:

$$\begin{aligned}
self :: n &\equiv \epsilon :: n,\\
child :: n &\equiv \Downarrow :: n,\\
descendant :: n &\equiv \Downarrow^* :: * / \Downarrow :: n,\\
descendant\text{-}or\text{-}self :: n &\equiv \Downarrow^* :: n,\\
parent :: n &\equiv \Uparrow :: n,\\
ancestor :: n &\equiv \Uparrow^* :: * / \Uparrow :: n,\\
ancestor\text{-}or\text{-}self :: n &\equiv \Uparrow^* :: n,\\
following\text{-}sibling :: n &\equiv \Rightarrow^* :: * / \Rightarrow :: n,\\
following :: n &\equiv \Uparrow^* :: * / \Rightarrow :: * / \Rightarrow^* :: * / \Downarrow^* :: n,\\
following\text{-}sibling :: n &\equiv \Leftarrow^* :: * / \Leftarrow :: n,\\
preceding :: n &\equiv \Uparrow^* :: * / \Leftarrow :: * / \Leftarrow^* :: * / \Downarrow^* :: n.
\end{aligned}$$

As in the standard, the child axis is the default axis, i.e. if the axis is not specified, then the $\Downarrow$ axis is assumed. This allows us to write a/b instead of the expression $\Downarrow :: a/ \Downarrow :: b$.

We also identify the following subclasses of queries:

- $\mathcal{X}C$ – no union, axes restricted to $\Downarrow$, $\Downarrow^*$, and ϵ, no disjunction and negation in the filter expressions, and no join conditions.
- $\mathcal{X}C(\vee)$ – $\mathcal{X}C$ with union and disjunction.
- $\mathcal{X}C(\neg)$ – $\mathcal{X}C$ with negation.
- $\mathcal{X}C(=)$ – $\mathcal{X}C$ with join conditions.
- $\mathcal{X}C(\Uparrow)$ – $\mathcal{X}C$ with the ascending axes $\Uparrow$ and $\Uparrow^*$.
- $\mathcal{X}C(\Leftarrow)$ – $\mathcal{X}C$ with the sliding axes $\Leftarrow$ and $\Leftarrow^*$.
- $\mathcal{X}C(\Rightarrow)$ – $\mathcal{X}C$ with the sliding axes $\Rightarrow$ and $\Rightarrow^*$.

6.3.1 Tree facts

We use the standard semantics of XPath queries [W3C99, Mar04], defined in a way geared toward the computation of valid query answers. The main notion we use is that of a *tree fact* which is a triple (x, Q, y), where y is an object (a node or a text label) reachable from a node x with a query Q. We distinguish *basic* tree facts which capture the structure of the tree and its textual contents.

Definition 6.13 (Basic tree facts) Given an tree T, the set of *basic facts* of T consists of the following elements:

- $(n, \epsilon :: X, n)$ for any node n of T whose label is X;
- $(n, \epsilon :: \mathit{text}(), t)$ for any text node n of T whose text label is t (in addition to the fact above);
- $(p, \Downarrow :: *, n)$ and $(n, \Uparrow :: *, p)$ for any node p of T and any child n of p;
- $(n_1, \Rightarrow :: *, n_2)$ and $(n_2, \Leftarrow :: *, n_1)$ for any pair of nodes n_1 and n_2 of T such that n_2 is the immediate following sibling of n_1.

Other tree facts are derived using implications that follow the standard semantics of XPath. We present the implications in Figure 6.4.

$$\begin{aligned}
(x, p_1 \cup \cdots \cup p_n, y) &\leftarrow p_i && \text{for } i \in \{1, \ldots, n\},\\
(x, s_1/\cdots/s_m, y) &\leftarrow (x, s_1/\cdots/s_i, z) \wedge (z, s_{i+1}/\cdots/s_n, y) && \text{for } i \in \{1, \ldots, m\},\\
(x, a :: n, y) &\leftarrow (x, a :: *, y) \wedge (y, \epsilon :: n, y),\\
(x, a :: n[f], y) &\leftarrow (x, a :: n, y) \wedge (y, \epsilon :: *[f], y),\\
(x, a^* :: *, x) &\leftarrow (x, \epsilon :: *, x),\\
(x, a^* :: *, y) &\leftarrow (x, a :: *, z) \wedge (z, a^* :: *, y),\\
(x, \epsilon :: *, x) &\leftarrow (x, \epsilon :: X, x),\\
(x, \epsilon :: *[q], x) &\leftarrow (x, \epsilon :: *, x) \wedge (x, q, y),\\
(x, \epsilon :: *[q = str], x) &\leftarrow (x, \epsilon :: *, x) \wedge (x, q, str),\\
(x, \epsilon :: *[q_1 = q_2], x) &\leftarrow (x, \epsilon :: *, x) \wedge (x, q_1, y) \wedge (x, q_2, y),\\
(x, \epsilon :: *[\neg f], x) &\leftarrow \mathbf{not}(x, \epsilon :: *[f], x),\\
(x, \epsilon :: *[f_1 \wedge f_2], x) &\leftarrow (x, \epsilon :: *[f_1], x) \wedge (x, \epsilon :: *[f_k], x),\\
(x, \epsilon :: *[f_1 \vee f_2], x) &\leftarrow (x, \epsilon :: *[f_1], x),\\
(x, \epsilon :: *[f_1 \vee f_2], x) &\leftarrow (x, \epsilon :: *[f_2], x).
\end{aligned}$$

Figure 6.4: Fact derivation rules.

Definition 6.14 (Query answer) x is an *answer* to Q in T if (r, Q, x) can be derived for T, where r is the root node of T. We denote the set of all answers to Q in T with $QA(T, Q)$.

We note that when computing answers to a query Q we need to use only a fixed number of different derivation rules (which involve only subqueries of Q). If we use only positive queries, then the rules do not contain negation. Therefore, the derivation of other tree facts, similarly to negation-free Datalog programs, is a monotonic process i.e., adding new (basic) facts cannot invalidate facts derived previously. This observation allows us to construct Algorithm 6.1 for computing the set of all tree facts that are relevant to answering a positive query Q (facts that use subqueries of Q). For every node it adds to the set of basic facts of the node the facts obtained recursively for the children of the node. In every step we also add any facts that can be derived with the derivation rules for the query Q; this operation is denoted by the superscript $(\cdot)^Q$. Finally, we select from the computed set the answers to Q.

Algorithm 6.1 Computing $QA(T,Q)$.

```
function FACTS(T,Q)
  $T = X(T_1,\ldots,T_n)$
  $r, r_1, \ldots, r_n$ are the root nodes of $T, T_1, \ldots, T_n$ respectively.
1:    $\mathcal{F} = \begin{cases} \{(\{(r,\epsilon :: X, r),(r,\epsilon :: \mathit{text}(),t)\})^Q\} & \text{if } T \text{ is a text node } t,\\ \{(\{(r,\epsilon :: X, r)\})^Q\} & \text{otherwise.}\end{cases}$
2:    for $i \in \{1,\ldots,n\}$ do
3:      if $i > 1$ then
4:        $\mathcal{F} := (\mathcal{F} \cup \{(r_{i-1}, \Rightarrow :: *, r_i),(r_i, \Leftarrow :: *, r_{i-1})\})^Q$
5:      $\mathcal{F} := (\mathcal{F} \cup \{(r, \Downarrow :: *, r_i),(r_i, \Uparrow :: *, r)\})^Q$
6:      $\mathcal{F} := (\mathcal{F} \cup \text{FACTS}(T_i,Q))^Q$
7:    return $\mathcal{F}$
end function
body
  $QA(T,Q) = \{x | (r,Q,x) \in \text{FACTS}(T,Q)\}$
end body
```

6.3.2 Valid query answers

Definition 6.15 (Valid query answers) Given a DTD D, a query Q, and a document T, an object x (either a node or a text label) is a *valid answer* to Q in T w.r.t. D if and only if x is an answer to Q in every repair of T w.r.t. D. By $VQA(T,Q,D)$ we denote the set of all valid answers to Q in T w.r.t. D.

Computational complexity

Recall that the combined complexity of computing answers to XPath queries is PTIME [GKP02]. Therefore, when we study tractability of computing query answers we include the query in the input and assume the DTD to be fixed. Formally we characterize the computational complexity of the membership problem for the following set

$$\mathcal{D}_D = \{(x,Q,T) \,|\, x \text{ is a valid answer to } Q \text{ in } T \text{ w.r.t. } D\}.$$

6.4 Tractable case

We start by presenting an exponential algorithm that computes valid query answers to any positive query. Later, we show how in two steps modify the algorithm to compute valid query answers to $\mathcal{XC}$ queries in polynomial time.

Again for clarity we consider only inserting and deleting tree operations: the approach can be extended to modifying operations in a way analogous to trace graphs in Section 6.2.3.

6.4.1 Exponential algorithm

Algorithm 6.2 computes valid query answers by (recursively) constructing a collection of sets of facts that hold in repairs of the document. Basically, for every repairing path in the trace graph the algorithm constructs the set of tree facts present in the repairs represented by this path. These sets are constructed in an incremental fashion using a flooding technique: V stores visited nodes and $\mathbb{C}(v)$ stores a collection of sets of facts for all paths from q_0^0 to v. For the empty path we only include basic tree facts of the root node. For every traversed edge we *append* all sets in the collection corresponding to the edge: none for a Del edge; for $Read$ edge the collection of sets from the corresponding subtree; for $Ins\ Y$ the collection of sets of facts $\mathcal{M}_Y$ that are present in every minimal valid tree with root Y (computed with a simple algorithm). The *appending operation* $\uplus^r$ is union which also adds basic facts establishing parent-child relation (between r and the root node of the appended subtree) and basic facts establishing sibling relation (and order) among appended trees. In every step we also add any facts that can be derived with the derivation rules for the query Q; this operation is denoted by the superscript $(\cdot)^Q$.

Proposition 6.16 *The Algorithm 6.2 computes valid answers to a positive query Q.*

6.4.2 Certain facts

The first proposed modification moves the intersection from the very last moment of computing valid query answers to the place where the collection of sets computed for a whole (sub)tree is returned. The set of facts obtained by intersection of all sets corresponding to the repairs of a given tree is called *certain facts*. This approach is implemented in Algorithm 6.3.

Example 6.17 Recall the tree T_1 from Figure 6.1 and the DTD D_1 from Example 6.1. Consider the the query $Q_1 = \epsilon::\mathtt{C}/\Downarrow^*/text()$. The trace graph for T_1 and D_1 can be found

Algorithm 6.2 Computing $VQA(T, Q, D)$

```
function REPAIRED($T$,$D$,$Q$)
  $T = X(T_1, \ldots, T_n)$ with the root node $r$
  $M_{D(X)} = \langle \Sigma, S, q_0, \Delta, F \rangle$
  $in(v) \overset{\text{def}}{=} \{u \in U_T^* \mid u \longrightarrow v\}$ (* incoming neighbors *)
  precompute:
    $U_T^*$ – the trace graph of $T$.
    $\mathcal{M}_Y$ – collection of sets of tree facts from every minimal tree with the root label $Y \in \Sigma$.
    $\mathcal{M}_i := \text{REPAIRED}(T_i, D, Q)$ (for $i \in \{1, \ldots, n\}$).
1:     $V := \{q_0^0\}$
2:     $\mathbb{C}(q_0^0) := \begin{cases} \{(\{(r, \epsilon :: X, r), (r, \epsilon :: text(), t)\})^Q\} & \text{if } T \text{ is a text node } t, \\ \{(\{(r, \epsilon :: X, r)\})^Q\} & \text{otherwise.} \end{cases}$
3:     while $V \neq U_T^*$ do
4:       choose any $q^i \in U_T^*$ such that $in(q^i) \subseteq V$
5:       $V := V \cup \{q^i\}$
6:       $\mathbb{C}(q^i) := \varnothing$
7:       if $q^{i-1} \xrightarrow{Del} q^i \in U_T^*$ then $\mathbb{C}(q^i) := \mathbb{C}(q^{i-1})$
8:       for $p^i \xrightarrow{Ins\ Y} q^i \in U_T^*$ do
9:         $\mathbb{C}(q^i) := \mathbb{C}(q^i) \cup \{(C \uplus^r C_Y)^Q | C \in \mathbb{C}(p^i) \wedge C_Y \in \mathcal{M}_Y\}$
10:      for $p^{i-1} \xrightarrow{Read} q^i \in U_T^*$ do
11:        $\mathbb{C}(q^i) := \mathbb{C}(q^i) \cup \{(C \uplus^r C_i)^Q | C \in \mathbb{C}(p^{i-1}) \wedge C_i \in \mathcal{M}_i\}$
12:    return $\bigcup_{q \in F \wedge q^n \in U_T^*} \mathbb{C}(q^n)$
end function
body
  $VQA(T, Q, D) = \{x | (r, Q, x) \in \bigcap \text{REPAIRED}(T, D, Q)\}$
end body
```

in Figure 6.3.

$\mathbb{C}(q_0^0) = \{B_0\}$, where

$$B_0 = (\{(n_0, \epsilon :: \mathtt{C}, n_0), \})^{Q_1}.$$

$\mathbb{C}(q_1^1) = \{B_1\}$, where

$$B_1 = (B_0 \cup C_1 \cup \{(n_0, \Downarrow :: *, n_1)\})^{Q_1},$$

and C_1 is the set of certain facts for $\mathtt{A}(\mathtt{d})$

$$C_1 = (\{(n_1, \epsilon :: \mathtt{A}, n_1), (n_1, \Downarrow, n_2),$$

$$(n_2, \epsilon :: \mathtt{PCDATA}, n_2), \qquad (n_2, \epsilon :: text(), \mathtt{d})\})^{Q_1}.$$

Algorithm 6.3 Computing $VQA(T,Q,D)$

```
function CERTAIN($T$,$D$,$Q$)
  $T = X(T_1, \ldots, T_n)$ with the root node $r$
  $M_{D(X)} = \langle \Sigma, S, q_0, \Delta, F \rangle$
  $in(v) \overset{\text{def}}{=} \{u \in U_T^* \,|\, u \longrightarrow v\}$ (* incoming neighbors *)
  precompute:
    $U_T^*$ – the trace graph for $T$.
    $C_Y$ – tree facts common for every valid tree with the root label $Y \in \Sigma$.
    $C_i :=$ CERTAIN($T_i, D, Q$) (for $i \in \{1, \ldots, n\}$).
1:    $V := \{q_0^0\}$
2:    $\mathbb{C}(q_0^0) := \begin{cases} \{(\{(r, \epsilon :: X, r), (r, \epsilon :: text(), t)\})^Q\} & \text{if } T \text{ is a text node } t, \\ \{(\{(r, \epsilon :: X, r)\})^Q\} & \text{otherwise.} \end{cases}$
3:    while $V \neq U_T^*$ do
4:      choose any $q^i \in U_T^*$ such that $in(q^i) \subseteq V$
5:      $V := V \cup \{q^i\}$
6:      $\mathbb{C}(q^i) := \varnothing$
7:      if $q^{i-1} \xrightarrow{Del} q^i \in U_T^*$ then $\mathbb{C}(q^i) := \mathbb{C}(q^{i-1})$
8:      for $p^i \xrightarrow{Ins\ Y} q^i \in U_T^*$ do
9:        $\mathbb{C}(q^i) := \mathbb{C}(q^i) \cup \{(C \uplus^r C_Y)^Q | C \in \mathbb{C}(p^i)\}$
10:     for $p^{i-1} \xrightarrow{Read} q^i \in U_T^*$ do
11:       $\mathbb{C}(q^i) := \mathbb{C}(q^i) \cup \{(C \uplus^r C_i)^Q | C \in \mathbb{C}(p^{i-1})\}$
12:   return $\bigcap \left( \bigcup_{q^n \in U_T^*} \mathbb{C}(q^n) \right)$
end function
body
  $VQA(T,Q,D) = \{x | (r,Q,x) \in$ CERTAIN($T,D,Q$)$\}$
end body
```

$\mathbb{C}(q_2^0) = \{B_2\}$, where

$$B_2 = (B_1 \cup C_2 \cup \{(n_0, \Downarrow :: *, n_3), (n_3, \Leftarrow :: *, n_1)\})^{Q_1},$$

and C_2 is the set of certain facts for the second child (note that C_2 doesn't contain information about n_4)

$$C_2 = (\{(n_3, \epsilon :: \mathtt{B}, n_3)\})^{Q_1}.$$

$\mathbb{C}(q_1^2) = \{B_1, B_3\}$, where

$$B_3 = (B_1 \cup C_{\mathtt{A}} \cup \{(n_0, \Downarrow :: *, i_1), (i_1, \Leftarrow :: *, n_3)\})^{Q_1},$$

where $C_{\mathtt{A}}$ is the set of certain facts for every valid tree with the root label A (i_1 is a new node)

$$C_{\mathtt{A}} = (\{(i_1, \epsilon :: \mathtt{A}, i_1)\})^{Q_1}.$$

$\mathbb{C}(q_0^3) = \{B_2, B_4, B_5\}$, where

$$B_4 = (B_3 \cup C_3 \cup \{(n_0, \Downarrow :: *, n_5), (n_5, \Leftarrow :: *, i_1)\})^{Q_1},$$

$$B_5 = (B_1 \cup C_3 \cup \{(n_0, \Downarrow :: *, n_5), (n_5, \Leftarrow :: *, n_1)\})^{Q_1},$$

where C_3 is the set of certain facts for the third child

$$C_3 = (\{(n_5, \epsilon :: \mathtt{B}, n_5)\})^{Q_1}.$$

Now the set of certain facts for T_1 is $C = B_2 \cap B_4 \cap B_5$. Note that $(n_0, Q_1, \mathsf{d}) \in C$ but $(n_0, Q_1, \mathsf{e}) \notin C$. Hence

$$VQA(T_1, Q_1, D_1) = \{\mathsf{d}\}.$$

If we compare the valid answers with $QA(T_1, Q_1) = \{\mathsf{d}, \mathsf{e}\}$, we note that e has been removed. This is because D_1 does not allow any (text) nodes under B.

We recall from Example 6.11 that our framework allows more than one isomorphic repair. Therefore the set of valid answers to query $\Downarrow^*$::B in T_1 is empty. This is a consequence of our decision to query the invalid document without repairing it, and thus the valid answers have to be given in terms of the original document.

Lemma 6.18 *For any $Q \in \mathcal{XC}$, any DTD D, and any document T Algorithm 6.3 computes valid answers to Q in T w.r.t. D.*

Proof[Sketch] We prove the lemma for queries using $\Downarrow$ axis only and having no conditions, i.e. $p = Y_1/\cdots/Y_k$.

We prove by induction over the structure of the document T that $(x, p, y) \in \textsc{Cert}(T, D, Q)$ if and only if (x, p, y) is a fact of every repair of T w.r.t. D. For the base case (the document is a single node), the hypothesis is trivial. Now, we assume that the hypothesis hold for every subtree of T, in particular for $T_1, \ldots, T_n$.

$\boxed{\Rightarrow}$ Assume that $(x, p, y) \in \textsc{Cert}(T, D, Q)$.

If $x \neq r$, then it means that every repairing path involved reading (or inserting) the subtree that contains x. By induction hypothesis this fact belongs to every repair of the given subtree and because every repairing path involves including this subtree, also every repair of T has a repair of the subtree. Hence, (x, p, y) is a fact present in every repair of T as well.

Suppose that $x = r$. Then $(r, Y_1/\cdots/Y_k, y) \in \textsc{Cert}(T, D, Q)$ implies that on every repair path a child r' of r is included such that $(r', Y_2/\cdots/Y_k, y)$ is a certain fact of T' (where r' can be either one of the children of r in T or the root of a subtree inserted with an inserting edge). By induction hypothesis $(r', Y_1/\cdots/Y_k, y)$ belongs to every repair of the subtree T'. We note that although not every repair path has to include the same tree that realizes the path fragment $Y_2/\cdots/Y_k$, there is at least one such tree on every path. Hence, in every repair of T the fact $(r, Y_1/\cdots/Y_k, y)$ can be inferred.

$\boxed{\Leftarrow}$ If the fact (x, p, y) is present in every repair and $x \neq r$, then every repair must contain a subtree having x. Consequently every repair path must include this subtree, and by induction hypothesis this fact is in $Cert(T, D, Q)$.

If $x = r$, then in every repair there must be a node r' such that $(r', Y_2/\cdots/Y_k, y)$. Although the node r' does not have to be common for all repairs, it is common for repairs obtained with the same repairing path because the p consists of descending steps only. Therefore, on every repairing path there exists $(r', Y_2/\cdots/Y_k, y)$ and together with (r, Y_1, r') give allows to derive (r, p, y) on every repairing path.

□

6.4.3 Eager cut

The next step required to make the algorithm work in polynomial time for descending queries is the heuristic of *eager cut*:

Let B_1 and B_2 be two sets from $\mathbb{C}(v)$ for some vertex v. Suppose that $v \longrightarrow v'$ and the edge is labeled with an operation that appends a tree (either *Rep* or *Ins*). Let B_1' and B_2' be the sets for v' obtained from B_1 and B_2 resp. as described before. Instead of storing in $\mathbb{C}(v')$ both sets B_1' and B_2' we only store their intersection $B_1' \cap B_2'$.

For instance, using this rule in Example 6.17 we obtain: $\mathbb{C}(q_0^3) = \{B_2, B'_{4,5}\}$, where $B'_{4,5} = B_4 \cap B_5$.

Proof Take any two paths p_1 and p_2 from the vertex q_0^0 to a node v and let p' be the path from v' to an accepting state of U_T^*. Then a fact (x, p, y) on the paths $p_1 \cdot v \to v' \cdot p'$ and $p_2 \cdot v \to v' \cdot p'$ is contributed either on the subpath p' and the eager cut does not involve it or it is contributed in both of the subpaths $p_1 \cdot v \to v'$ and $p_2 \cdot v \to v'$ and then the intersection does not remove it. □

Algorithm 6.4 Computing $VQA(T, Q, D)$ with eager cut

```
function Eager($T$,$D$,$Q$)
  $T = X(T_1, \ldots, T_n)$ with the root node $r$
  $M_{D(X)} = \langle \Sigma, S, q_0, \Delta, F \rangle$
  $in(v) \overset{\text{def}}{=} \{u \in U_T^* \mid u \longrightarrow v\}$ (* incoming neighbors *)
  precompute:
    $U_T^*$ – the trace graph for $T$.
    $C_Y$ – tree facts common for every valid tree with the root label $Y \in \Sigma$.
    $C_i :=$ Eager$(T_i, D, Q)$ (for $i \in \{1, \ldots, n\}$).
1:    $V := \{q_0^0\}$
2:    $\mathbb{C}(q_0^0) := \begin{cases} \{(\{(r, \epsilon :: X, r), (r, \epsilon :: text(), t)\})^Q\} & \text{if } T \text{ is a text node } t, \\ \{(\{(r, \epsilon :: X, r)\})^Q\} & \text{otherwise.} \end{cases}$
3:    while $V \neq U_T^*$ do
4:      choose any $q^i \in U_T^*$ such that $in(q^i) \subseteq V$
5:      $V := V \cup \{q^i\}$
6:      $\mathbb{C}(q^i) := \varnothing$
7:      if $q^{i-1} \xrightarrow{Del} q^i \in U_T^*$ then $\mathbb{C}(q^i) := \mathbb{C}(q^{i-1})$
8:      for $p^i \xrightarrow{Ins\ Y} q^i \in U_T^*$ do
9:        $\mathbb{C}(q^i) := \mathbb{C}(q^i) \cup \bigcap\{(C \uplus^r C_Y)^Q | C \in \mathbb{C}(p^i)\}$
10:     for $p^{i-1} \xrightarrow{Read} q^i \in U_T^*$ do
11:       $\mathbb{C}(q^i) := \mathbb{C}(q^i) \cup \bigcap\{(C \uplus^r C_i)^Q | C \in \mathbb{C}(p^{i-1})\}$
12:   return $\bigcap \left( \bigcup_{q^n \in U_T^*} \mathbb{C}(q^n) \right)$
end function
body
  $VQA(T, Q, D) = \{x | (r, Q, x) \in$ Eager$(T, D, Q)\}$
end body
```

Using simple induction over the number of the column, we can easily prove that in Algorithm 6.4 the number of sets stored in $\mathbb{C}(q^i)$ is $O(i \times |S| \times |\Sigma|)$.

Theorem 6.19 *Algorithm 6.4 computes valid answers to $\mathcal{XC}$ queries in polynomial time.*

6.5 Intractable cases

In this section we show that $\mathcal{X}C$ is a maximal class of queries for which the problem of consistent query answers is tractable: adding any of the additional features leads to intractability. First we state the upper bound of computing consistent query answers.

Lemma 6.20 *Valid query answering for $\mathcal{X}Core$ is in coNP.*

Proof For a given DTD D, a given document T, a given query q, and a given object x we use U_T^* to nondeterministically construct a repair T' such that x is *not* an answer to q in T'. $\square$

Theorem 6.21 *Valid query answering for $\mathcal{X}C(\neg)$ and $\mathcal{X}C(\vee)$ is coNP-complete.*

Proof We show coNP-hardness by reducing the complement of SAT to $\mathcal{D}_{D^\vee}$ for the following DTD:

$$D^\vee(\mathtt{A}) = \mathtt{V}^*, \qquad D^\vee(\mathtt{T}) = \mathtt{PCDATA},$$
$$D^\vee(\mathtt{V}) = \mathtt{T} + \mathtt{F}, \qquad D^\vee(\mathtt{F}) = \mathtt{PCDATA}.$$

Take any CNF formula $\varphi = c_1 \wedge \ldots \wedge c_k$ over the set of variables $\{x_1, \ldots, x_n\}$, where $c_j = l_{j,1} \vee \ldots \vee l_{j,m_j}$ is a clause of m_j literals for every $j \in \{1, \ldots, s\}$. We construct the following trees:

- for every $i \in \{1, \ldots, n\}$ by V_i we denote the tree $\mathtt{V}(\mathtt{T}(i), \mathtt{F}(i))$ whose root node is v_i;
- the tree $T_\varphi^\vee$ is $\mathtt{A}(V_1, \ldots, V_n)$ and the root of $T_\varphi^\vee$ is r.

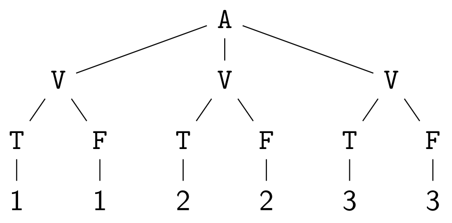
We construct the following queries:

- for a literal $l_{j,k}$ we construct the filter expression $\lambda_{j,k} = \mathtt{V}[\mathtt{F} = p]$ if $l_{j,k} = x_p$ and $\lambda_{j,k} = \mathtt{V}[\mathtt{T} = p]$ if $l_{j,k} = \neg x_p$;
- for a clause c_i we construct the filter expression $d_i = \lambda_{i,1} \wedge \ldots \wedge \lambda_{i,m_i}$;
- the query $Q_\varphi^\vee$ is $\epsilon :: *[d_1] \cup \ldots \cup \epsilon :: *[d_k]$;
- the query $Q_\varphi^{\vee\prime}$ is $\epsilon :: *[d_1 \vee \ldots \vee d_k]$;

- the query $Q^{\neg}_{\varphi}$ is $\epsilon :: *[\neg(\neg d_1 \wedge \ldots \wedge \neg d_k)]$.

Obviously, the queries $Q^{\vee}_{\varphi}$, $Q^{\vee\prime}_{\varphi}$, and $Q^{\neg}_{\varphi}$ are equivalent. We also observe that $Q^{\vee\prime}_{\varphi}, Q^{\vee}_{\varphi} \in \mathcal{XC}(\vee)$ and $Q^{\neg}_{\varphi} \in \mathcal{XC}(\neg)$.

Figure 6.5 contains an example of the reduction for $\varphi = (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3)$.



$$Q^{\vee}_{\varphi} = \epsilon :: *\big[\mathtt{V}[\mathtt{T} = 1] \wedge \mathtt{V}[\mathtt{F} = 2]\big] \cup \epsilon :: *\big[\mathtt{V}[\mathtt{F} = 1] \wedge \mathtt{V}[\mathtt{F} = 3]\big].$$

Figure 6.5: $T^{\vee}_{\varphi}$ and $Q^{\vee}_{\varphi}$ for $\varphi = (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3)$.

We observe a one-to-one correspondence between the repairs of $T^{\vee}_{\varphi}$ and the valuations of $x_1, \ldots, x_n$. For a valuation V the corresponding tree is $\mathtt{A}(V'_1, \ldots, V'_n)$ where $V'_i = \mathtt{V}(\mathtt{T}(i))$ if $V(x_i) = \mathit{true}$ and $V'_i = \mathtt{V}(\mathtt{F}(i))$ if $V(x_i) = \mathit{false}$ for $i \in \{1, \ldots, n\}$. Given a repair $T' \in \mathit{Rep}_D(T^{\vee}_{\varphi})$ corresponding to the valuation V, we observe that:

- $(r, \epsilon :: *[\lambda_{j,l}], r)$ is a fact of T' if and only if $V \not\models l_{j,k}$ for every $j \in \{1, \ldots, s\}$ and every $k \in \{1, \ldots, m_j\}$;
- $(r, \epsilon :: *[d_j], r)$ is a fact of T' if and only if $V \not\models c_j$ for every $j \in \{1, \ldots, s\}$;
- $(r, Q^{\vee}_{\varphi}, r)$ is a fact of T' if and only if $V \not\models \varphi$;

Because there is one-to-one correspondence between the repairs of $T^{\vee}_{\varphi}$ and the valuations of φ we have that r is valid answer to $Q^{\vee}_{\varphi}$ ($Q^{\vee\prime}_{\varphi}$, and $Q^{\neg}_{\varphi}$) in $T^{\vee}_{\varphi}$ w.r.t. $D_{\vee}$ if and only if $\varphi \notin$ SAT. $\square$

Theorem 6.22 *Valid query answering for $\mathcal{XC}(\Uparrow)$ is coNP-complete.*

Proof We show coNP-hardness by reducing the complement of SAT to $\mathcal{D}_{D^{\Uparrow}}$ for the following DTD:

$$D^{\Uparrow}(\mathtt{A}) = (\mathtt{A} \cdot \mathtt{V}) + \mathtt{C}^*, \qquad D^{\Uparrow}(\mathtt{C}) = \mathtt{V} \cdot \mathtt{V} \cdot \mathtt{B}^*, \qquad D^{\Uparrow}(\mathtt{T}) = \epsilon,$$

$$D^{\Uparrow}(\mathtt{V}) = \mathtt{T} + \mathtt{F}, \qquad D^{\Uparrow}(\mathtt{B}) = \mathtt{B} + \mathtt{PCDATA}, \qquad D^{\Uparrow}(\mathtt{F}) = \epsilon.$$

Take any CNF formula $\varphi = c_1 \wedge \ldots \wedge c_k$ over the set of variables $\{x_1, \ldots, x_n\}$, where $c_j = l_{j,1} \vee \ldots \vee l_{j,m_j}$ is a clause of m_j literals for every $j \in \{1, \ldots, k\}$. We construct the following trees:

- for any $i \in \{1, \ldots, n\}$ and any $j \in \{1, \ldots, k\}$ we construct two trees:

$$P_{i,j} = \underbrace{\mathtt{B}(\ldots\mathtt{B}(}_{w} i) \ldots), \quad \text{where } w = \begin{cases} i & \text{if } c_j \text{ uses } x_i, \\ n+2 & \text{otherwise,} \end{cases}$$

  and

$$N_{i,j} = \underbrace{\mathtt{B}(\ldots\mathtt{B}(}_{w} \sim i) \ldots), \quad \text{where } w = \begin{cases} i & \text{if } c_j \text{ uses } \neg x_i, \\ n+2 & \text{otherwise;} \end{cases}$$

- for any $j \in \{1, \ldots, k\}$ we construct the tree $C_j = \mathtt{C}(\mathtt{V}(\mathtt{T}), \mathtt{V}(\mathtt{F}), P_{1,j}, \ldots, P_{n,j}, N_{1,j}, \ldots, N_{n,j})$ whose root node is d_j;
- we construct the tree $A_{n+1} = \mathtt{A}(C_1, \ldots, C_k)$ and recursively $A_i = \mathtt{A}(A_{i+1}, \mathtt{V}(\mathtt{T}, \mathtt{F}))$ for $i \in \{n, \ldots, 1\}$;
- by $T^{\Uparrow}_{\varphi}$ we denote the tree A_1 whose root node is r.

We construct the following queries:

- for every $i \in \{1, \ldots, n\}$ we construct two filter expressions:

$$\omega_i = \Downarrow^* :: \mathtt{B}[text() = i] / \underbrace{\Uparrow :: * / \cdots / \Uparrow :: *}_{n+2} /\mathtt{V}/\mathtt{F},$$

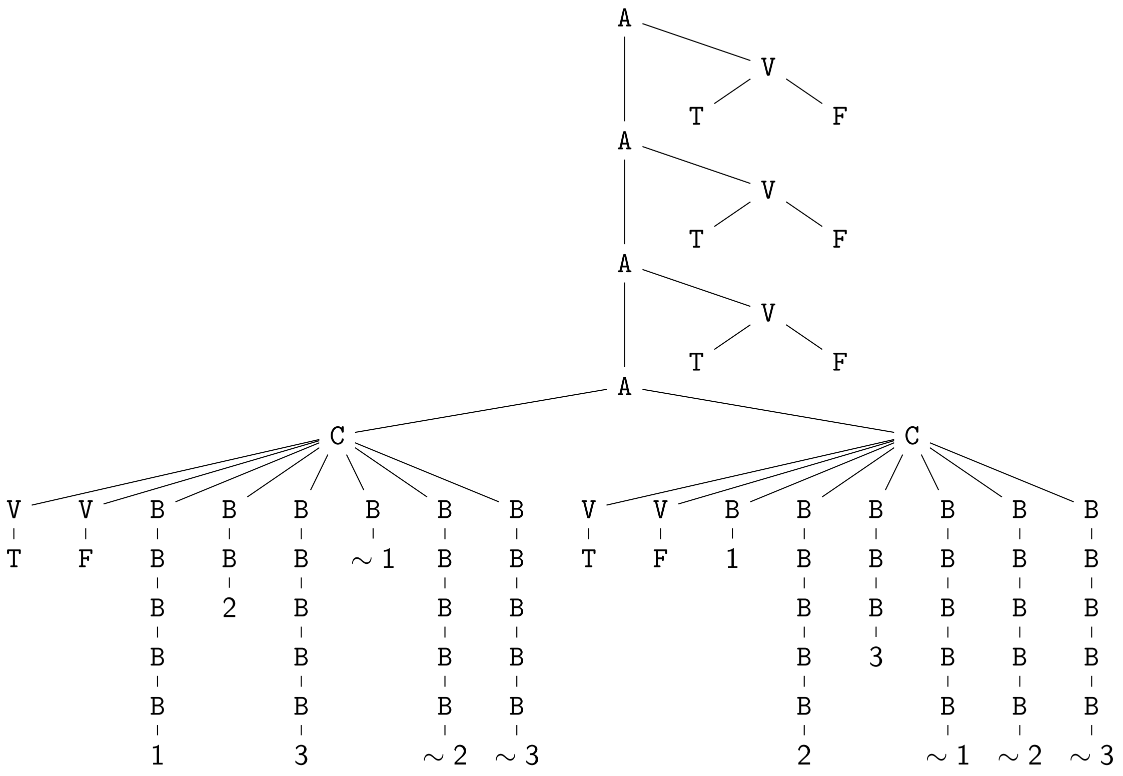
and

$$\bar{\omega}_i = \Downarrow^* :: \mathtt{B}[text() =\sim i]/\underbrace{\Uparrow :: */ \cdots / \Uparrow :: *}_{n+2}/\mathtt{V}/\mathtt{T};$$

- the following query

$$Q_\varphi^{\Uparrow} = \epsilon :: *\big[\Downarrow^* :: \mathtt{C}[\omega_1 \wedge \ldots \wedge \omega_n \wedge \bar{\omega}_1 \wedge \ldots \wedge \bar{\omega}_n]\big].$$

Figure 6.6 contains an example of the reduction for $\varphi = (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3)$. We observe



$$\begin{aligned} Q_\varphi^{\Uparrow} = \epsilon :: *\big[\Downarrow^* :: \mathtt{C}[&\Downarrow^* :: \mathtt{B}[text() = 1]/\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\mathtt{V}/\mathtt{F} \wedge \\ &\Downarrow^* :: \mathtt{B}[text() = 2]/\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\mathtt{V}/\mathtt{F} \wedge \\ &\Downarrow^* :: \mathtt{B}[text() = 3]/\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\mathtt{V}/\mathtt{F} \wedge \\ &\Downarrow^* :: \mathtt{B}[text() =\sim 1]/\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\mathtt{V}/\mathtt{T} \wedge \\ &\Downarrow^* :: \mathtt{B}[text() =\sim 2]/\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\mathtt{V}/\mathtt{T} \wedge \\ &\Downarrow^* :: \mathtt{B}[text() =\sim 3]/\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\Uparrow :: */\mathtt{V}/\mathtt{T}]\big]. \end{aligned}$$

Figure 6.6: $T_\varphi^{\Uparrow}$ and $Q_\varphi^{\Uparrow}$ for $\varphi = (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3)$.

that there is a one-to-one correspondence between the repairs of $T_\varphi^{\Uparrow}$ and the valuations of $x_1, \ldots, x_n$. For a valuation V the corresponding repair is constructed as follows:

- $A'_{n+1} = \mathtt{A}(C_1, \ldots, C_k)$;
- for $i \in \{n, \ldots, 1\}$ we construct $A'_i = \mathtt{A}(A'_{i+1}, W_i)$, where $W_i = \mathtt{V}(\mathtt{T})$ if $V(x_i) = \mathit{true}$ and $W_i = \mathtt{V}(\mathtt{F})$ if $V(x_i) = \mathit{false}$;
- the repair T' is A'_1.

Given a repair T' and the corresponding valuation V we observe that:

- for every $j \in \{1, \ldots, k\}$ and every $i \in \{1, \ldots, n\}$ the clause c_j does not use x_i or c_j uses x_i and $V \not\models x_i$ if and only if $(d_j, \epsilon :: *[\omega_i], d_j)$ is a fact of T';
- for every $j \in \{1, \ldots, k\}$ and every $i \in \{1, \ldots, n\}$ the clause c_j does not use $\neg x_i$ or c_j uses $\neg x_i$ and $V \not\models \neg x_i$ if and only if $(d_j, \epsilon :: *[\bar{\omega}_i], d_j)$ is a fact of T';
- for every $j \in \{1, \ldots, k\}$ the clause c_j is not satisfied by V if and only if $(d_j, \epsilon :: *[\omega_1 \wedge \bar{\omega}_1 \wedge \ldots \wedge \omega_n \wedge \bar{\omega}_n], d_j)$ is a fact of T';
- $V \not\models \varphi$ if and only if $(r, Q_\varphi^{\Uparrow}, r)$ is a fact of T'.

Because of one-to-one correspondence between the repairs of $T_\varphi^{\Uparrow}$ and the valuations of $x_1, \ldots, x_n$, $\varphi \notin$ SAT if and only if r is valid answer to $Q_\varphi^{\Uparrow}$ in $T_\varphi^{\Uparrow}$ w.r.t. D. □

Theorem 6.23 *Valid query answering for $\mathcal{X}C(=)$ is coNP-complete.*

Proof We show coNP-hardness by reducing the complement of 3SAT to $\mathcal{D}_{D^=}$ for the following DTD:

$$D^=(\mathtt{C}) = (\mathtt{C} \cdot \mathtt{N}_1 \cdot \mathtt{N}_2 \cdot \mathtt{N}_3) + \mathtt{B}^*, \qquad D^=(\mathtt{N}_1) = \mathtt{PCDATA},$$
$$D^=(\mathtt{B}) = \mathtt{V}^*, \qquad D^=(\mathtt{N}_2) = \mathtt{PCDATA},$$
$$D^=(\mathtt{V}) = \mathtt{PCDATA}, \qquad D^=(\mathtt{N}_3) = \mathtt{PCDATA}.$$

Take any 3CNF formula $\varphi = c_1 \wedge \ldots \wedge c_k$ over the set of variables $\{x_1, \ldots, x_n\}$, where $c_j = l_{j,1} \vee l_{j,2} \vee l_{j,3}$ is a clause of 3 literals for every $j \in \{1, \ldots, k\}$. We construct the following trees:

- for every $i \in \{1, \ldots, n\}$ the tree $B_i = \mathtt{B}(\mathtt{V}(i), \mathtt{V}(\sim i))$;
- for every $j \in \{1, \ldots, k\}$ and every $l \in \{1, 2, 3\}$ the tree $L_{j,l} = \mathtt{N}_l(\sim p)$ if $l_{j,l} = x_p$ or the tree $L_{j,l} = \mathtt{N}_l(p)$ if $l_{j,l} = \neg x_p$;
- the tree $C_{k+1} = \mathtt{C}(B_1, \ldots, B_n)$ and for $j \in \{k, \ldots, 1\}$ recursively the trees $C_j = \mathtt{C}(C_{j+1}, L_{j,1}, L_{j,2}, L_{j,3})$ whose root node is d_j;
- by $T_\varphi^=$ we denote the tree C_1.

We construct the following query:

$$\begin{aligned} Q^= = \epsilon\big[\Downarrow^* :: \mathtt{C}[&\mathtt{N}_1/text() = \Downarrow^* :: \mathtt{V}/text() \wedge \\ &\mathtt{N}_2/text() = \Downarrow^* :: \mathtt{V}/text() \wedge \\ &\mathtt{N}_3/text() = \Downarrow^* :: \mathtt{V}/text()]\big]. \end{aligned}$$

Figure 6.7 contains an example of reduction of $\varphi = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee x_4 \vee x_5)$. We note that the query is not dependent on the formula.

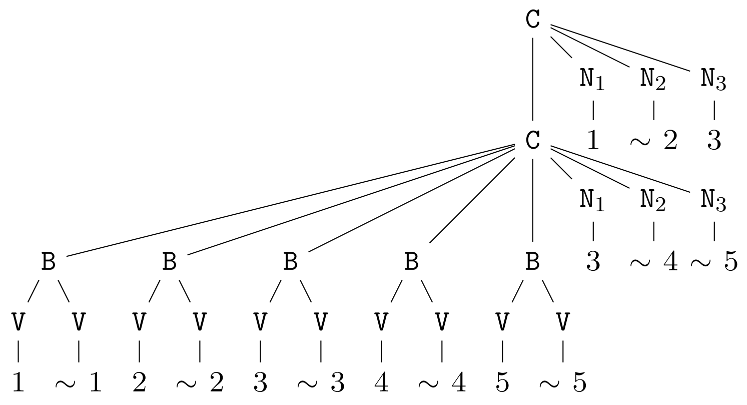


Figure 6.7: $T_\varphi^=$ for $\varphi = (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee x_4 \vee x_5)$.

We observe that there is a one-to-one correspondence between the repairs of $T_\varphi^=$ and the valuations of $x_1 \ldots, x_n$. For a validation V we construct the corresponding repairs as follows:

- $C'_{k+1} = \mathtt{C}(W_1, \ldots, W_n)$, where $W_i = \mathtt{B}(\mathtt{V}(i))$ if $V(x_i) = true$ and $W_i = \mathtt{B}(\mathtt{V}(\sim i))$ if $V(x_i) = false$ for $i \in \{1, \ldots, n\}$;

- for $j \in \{k, \ldots, 1\}$ we construct $C'_j = \mathtt{C}(C'_{j+1}, L_{j,1}, L_{j,2}, L_{j,3})$;
- the repair T' is C'_1.

Given a repair T' and the corresponding valuation V we observe that:

- for any $j \in \{1, \ldots, k\}$ and any $s \in \{1, 2, 3\}$ the literal $l_{j,s}$ is not satisfied by V if and only if $(d_j, \epsilon :: *[\lambda_s], d_j)$ is a fact of T', where $\lambda_s = (\mathtt{N}_s/\mathit{text}() = \Downarrow^* :: \mathtt{V}/\mathit{text}())$;
- for any $j \in \{1, \ldots, k\}$ the clause c_j is not satisfied by V if and only if $(d_j, \epsilon :: *[\lambda_1 \wedge \lambda_2 \wedge \lambda_3], d_j)$ is a fact of T';
- $V \not\models \varphi$ if and only if $(r, Q^{=}, r)$ is a fact of T'.

Because of the one-to-one correspondence between the repairs of $T^{=}_{\varphi}$ and the valuations of $x_1, \ldots, x_n$, $\varphi \notin$ 3SAT if and only if r is a valid answer to $Q^{=}$ in $T^{=}_{\varphi}$ w.r.t. $D^{=}$. □

Theorem 6.24 *Valid query answering for $\mathcal{XC}(\Leftarrow)$ is coNP-complete.*

Proof We show coNP-hardness by reducing SAT to $\mathcal{D}_D$ for the following DTD D:

$$D^{\Leftarrow}(\mathtt{A}) = \mathtt{C}, \qquad D^{\Leftarrow}(\mathtt{P}) = \mathtt{PCDATA},$$

$$D^{\Leftarrow}(\mathtt{C}) = \mathtt{B} \cdot \mathtt{A} \cdot \mathtt{P}^* \cdot \mathtt{A}^* \cdot \mathtt{P}^* + \mathtt{V}^*, \qquad D^{\Leftarrow}(\mathtt{T}) = \mathtt{PCDATA},$$

$$D^{\Leftarrow}(\mathtt{V}) = \mathtt{T} + \mathtt{F} \qquad D^{\Leftarrow}(\mathtt{F}) = \mathtt{PCDATA}.$$

We take any CNF formula $\varphi = c_1 \wedge \ldots \wedge c_k$ over a set of variables $\{x_1, \ldots, x_n\}$, where $c_j = l_{j,1} \vee \ldots \vee l_{j,m_j}$ is a clause of m_i literals for every $j \in \{1, \ldots, k\}$. We construct the following trees:

- for $i \in \{1, \ldots, n\}$ we construct

$$V_i = \mathtt{V}(\mathtt{T}(i), \mathtt{F}(i)),$$

$$V_i^P = \mathtt{V}(\mathtt{T}(i)),$$

$$V_i^N = \mathtt{V}(\mathtt{F}(i));$$

- we construct

$$C_{2k} = \mathtt{C}(V_1, \ldots, V_n),$$
$$C_{2k}^P = \mathtt{C}(V_1^P, \ldots, V_n^P),$$
$$C_{2k}^N = \mathtt{C}(V_1^N, \ldots, V_n^N);$$

- for $j \in \{2k-1, \ldots, k+1\}$ we construct

$$C_j = \mathtt{C}(\mathtt{B}, \mathtt{A}(C_{j+1})),$$
$$C_j^P = \mathtt{C}(\mathtt{B}, \mathtt{A}(C_{j+1}^P)),$$
$$C_j^N = \mathtt{C}(\mathtt{B}, \mathtt{A}(C_{j+1}^N));$$

- we construct $A^P = \mathtt{A}(C_{k+1}^P)$ and $A^N = \mathtt{A}(C_{k+1}^N)$
- for $j \in \{k, \ldots, 1\}$ we construct

$$C_j = \mathtt{C}(\mathtt{B}, \mathtt{A}(C_{j+1}), L_{j,1}, \ldots, L_{j,m_j}, A^P, A^N, U_1, \ldots, U_{k_1}, \bar{U}_1, \ldots, \bar{U}_{k_2}),$$

  where:

  - $L_{j,s} = \mathtt{P}(p)$ if $l_{j,s} = x_p$ and $L_{j,s} = \mathtt{P}(\sim p)$ if $l_{j,s} = \neg x_p$,
  - k_1 is the number of positive literals not used in the clause c_j,
  - for $s \in \{1, \ldots, k_1\}$ $U_s = \mathtt{P}(p)$, where p is the index of s-th consecutive positive literal not used in c_j,
  - k_2 is the number of negative literals not used in the clause c_j,
  - for $s \in \{1, \ldots, k_2\}$ $\bar{U}_s = \mathtt{P}(\sim p)$, where p is the index of s-th consecutive negative literal not used in c_j,
  - the root node of C_j is d_j.

- the tree $T_\varphi^\Leftarrow$ is C_1.

Figure 6.8 contains an example of document constructed for $\varphi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$.

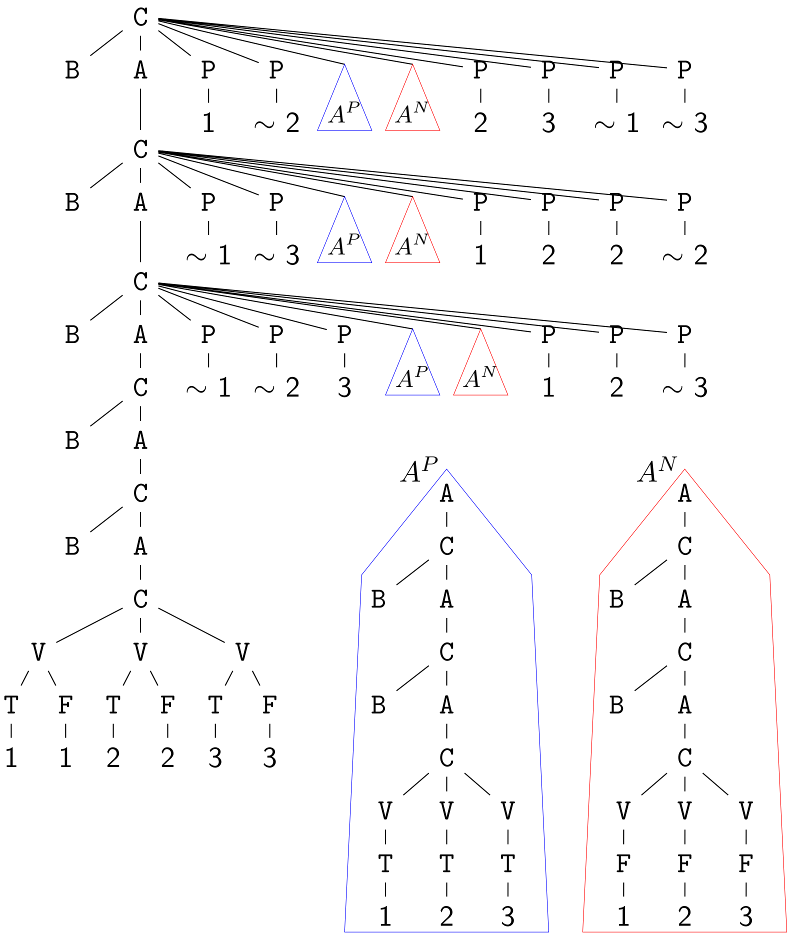


Figure 6.8: $T_\varphi^{\Leftarrow}$ for $\varphi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$.

We construct the following queries:

- for $i \in \{1, \dots, n\}$ we construct

$$\omega_i = \mathtt{P}[text() = i \wedge \Leftarrow^* :: \mathtt{A}\,/\,\underbrace{\mathtt{C}\,/\,\mathtt{A}[\Leftarrow :: \mathtt{B}]\,/\ldots\mathtt{C}\,/\,\mathtt{A}[\Leftarrow :: \mathtt{B}]\,/}_{k-1} \Downarrow^* :: \mathtt{C}\,/\,\mathtt{V}\,/\,\mathtt{F} = i];$$

- for $i \in \{1, \dots, n\}$ we construct

$$\bar{\omega}_i = \mathtt{P}[text() =\sim i \wedge \Leftarrow^* :: \mathtt{A}\,/\,\underbrace{\mathtt{C}\,/\,\mathtt{A}[\Leftarrow :: \mathtt{B}]\,/\ldots\mathtt{C}\,/\,\mathtt{A}[\Leftarrow :: \mathtt{B}]\,/}_{k-1} \Downarrow^* :: \mathtt{C}\,/\,\mathtt{V}\,/\,\mathtt{T} = i];$$

- the query
$$Q_\varphi^{\Uparrow} = \Downarrow^* :: \mathtt{C}[\omega_1 \wedge \ldots \wedge \omega_n \wedge \bar{\omega}_1 \wedge \ldots \wedge \bar{\omega}_n].$$

Figure 6.9 contains the query $Q_\varphi^{\Leftarrow}$ obtained for the formula $\varphi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$.

$$\begin{aligned}
Q_\varphi^{\Leftarrow} = \Downarrow^* :: \mathtt{C}\big[&\mathtt{P}[\mathit{text}() = 1 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{F} = 1] \wedge \\
&\mathtt{P}[\mathit{text}() = 2 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{F} = 2] \wedge \\
&\mathtt{P}[\mathit{text}() = 3 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{F} = 3] \wedge \\
&\mathtt{P}[\mathit{text}() = 4 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{F} = 4] \wedge \\
&\mathtt{P}[\mathit{text}() = 5 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{F} = 5] \wedge \\
&\mathtt{P}[\mathit{text}() = 6 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{F} = 6] \wedge \\
&\mathtt{P}[\mathit{text}() =\sim 1 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{T} = 1] \wedge \\
&\mathtt{P}[\mathit{text}() =\sim 2 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{T} = 2] \wedge \\
&\mathtt{P}[\mathit{text}() =\sim 3 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{T} = 3] \wedge \\
&\mathtt{P}[\mathit{text}() =\sim 4 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{T} = 4] \wedge \\
&\mathtt{P}[\mathit{text}() =\sim 5 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{T} = 5] \wedge \\
&\mathtt{P}[\mathit{text}() =\sim 6 \wedge \Leftarrow^* :: \mathtt{A} / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \mathtt{C} / \mathtt{A}[\Leftarrow :: \mathtt{B}] / \Downarrow^* :: \mathtt{C} / \mathtt{V} / \mathtt{T} = 6]\big].
\end{aligned}$$

Figure 6.9: Query $Q_\varphi^{\Leftarrow}$ for $\varphi = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$.

We observe that there is one-to-one correspondence between the repairs of $T_\varphi^{\Leftarrow}$ w.r.t. $D^{\Uparrow}$ and the valuations of $x_1, \ldots, x_n$. For a valuation V the repair T' is constructed like the tree $T_\varphi^{\Leftarrow}$ except that for $i \in \{1, \ldots, n\}$ instead of V_i we use V_i^P if $V(x_i) = \mathit{true}$ and V_i^N if $V(x_i) = \mathit{false}$.

Given a repair T' and the corresponding valuation V we observe that:

- for every $j \in \{1, \ldots, k\}$ and every $i \in \{1, \ldots, n\}$ the clause c_j does not use x_i or c_j uses x_i and $V \not\models x_i$ if and only if $(d_j, \epsilon :: *[\omega_i], d_j)$ is a fact of T';
- for every $j \in \{1, \ldots, k\}$ and every $i \in \{1, \ldots, n\}$ the clause c_j does not use $\neg x_i$ or c_j uses $\neg x_i$ and $V \not\models \neg x_i$ if and only if $(d_j, \epsilon :: *[\bar{\omega}_i], d_j)$ is a fact of T';
- for every $j \in \{1, \ldots, k\}$ the clause c_j is not satisfied by V if and only if $(c_j, \epsilon :: *[\omega_1 \wedge \ldots \wedge \omega_n \wedge \bar{\omega}_1 \wedge \ldots \wedge \bar{\omega}_n], c_j)$ is a fact of T';
- $V \not\models \varphi$ if and only if $(r, Q_\varphi^{\Leftarrow}, r)$ is a fact of T'.

Because of one-to-one correspondence between the repairs of $T_\varphi^{\Leftarrow}$ and the valuations of $x_1, \ldots, x_n$, $\varphi \notin \text{SAT}$ if and only if r is a valid answer to $Q_\varphi^{\Leftarrow}$ in $T_\varphi^{\Leftarrow}$ w.r.t. $D^{\Leftarrow}$. $\square$

We observe that if in the proof above we take a mirror image of the tree $T_\varphi^{\Leftarrow}$ and in the query $Q_\varphi^{\Leftarrow}$ change $\Leftarrow$ to $\Rightarrow$, we obtain a reduction and a coNP-completeness proof for $\mathcal{XC}(\Rightarrow)$.

Corollary 6.25 *Valid query answering for $\mathcal{X}C(\Rightarrow)$ is coNP-complete.*

6.6 Experimental evaluation

The main goal of our experiments was to validate our approach and to discover possible limitations to be addressed in the future. Because flat documents (documents of bounded height) are very common among XML repositories, we tested the behavior of our solutions on flat documents.

Data sets To observe the impact of the document size on the performance of algorithms we first randomly generated a valid document. Next, we introduced the violations of validity to a document by removing and inserting randomly chosen nodes. To measure the validity violations of a document T we use the *invalidity ratio* $\mathit{dist}(T, D)/|T|$.

For most of the experiments, we used the following DTD

$$D(\mathtt{A}) = \mathtt{B} \cdot \mathtt{C} \cdot \mathtt{D}, \qquad D(\mathtt{B}) = \mathtt{PCDATA} + \mathtt{A},$$
$$D(\mathtt{C}) = \mathtt{PCDATA} + \mathtt{A}, \qquad D(\mathtt{D}) = \mathtt{PCDATA} + \mathtt{A},$$

and the query $\epsilon :: \mathtt{A}/\mathtt{B}//\mathtt{C}/\mathit{text}()$. In experiments investigating the impact of the DTD size on the performance, we used a family of DTDs D_n, $n \geq 0$:

$$D_n(\mathtt{A}) = (\ldots((\mathtt{PCDATA} + \mathtt{A}_1) \cdot \mathtt{A}_2 + \mathtt{A}_3) \cdot \mathtt{A}_4 + \ldots \mathtt{A}_n)^*,$$
$$D_n(\mathtt{A}_i) = \mathtt{A}^*, \quad \text{for } i \in \{1, \ldots, n\}.$$

For documents generated for these DTDs we used a simple query $\Downarrow^*/\mathit{text}()$.

Implementation All compared algorithms were implemented in Java 5.0 and used common programming tools including: a *Pull* model parser (STaX), a representation of regular expressions and corresponding NDFAs, structures storing tree facts, and an implementation of derivation rules. For ease of implementation, we considered a restricted class of $\mathcal{XC}$ which involve only simple filter conditions (testing tag and text labels). We note that those queries are most commonly used in practice and the restrictions allow to compute standard answers to such queries in time linear in the size of the document.

Environment All tests were performed on an Intel Pentium Dual-Core 1.6GHz machine running Windows XP with 1.5 GB MB RAM and 80 GB hard drive. We repeated each test 5 times, discarded extreme readings, and took the average of the remaining ones.

6.6.1 Trace graph construction

In the first part of the experiments, we measured the time necessary to construct the trace graphs. We used algorithms MDist and Dist that constructed trace graphs (resp. with and without label modification) to compute the edit distance of the document to the DTD. As a base line we used the time necessary to parse the file containing the document (Parse). We also compared the performance of our algorithms with Validate which validates the database w.r.t. the DTD.

Figure 6.10 contains the results showing the impact of the document size on the performance of our algorithms. The results confirm our analysis: trace graphs are constructed in time linear in the size of the document. Note also that computation of the edit distance without using modifying operations introduces only small overhead over validation. Note also that the overhead of Validate over Parse is small as well, which suggests that the implementation of validation is efficient. We also observe that including modification operation in the model leads to a significant increase in the computation time.

Figure 6.11 shows that Dist takes the time quadratic in the size of the DTD. We note, however, that Validate behaves similarly and the overhead of Dist is small. Because in our approach we don't assume any particular properties of the automata used, we conjecture: that any technique that optimize the automata to efficiently validate XML documents

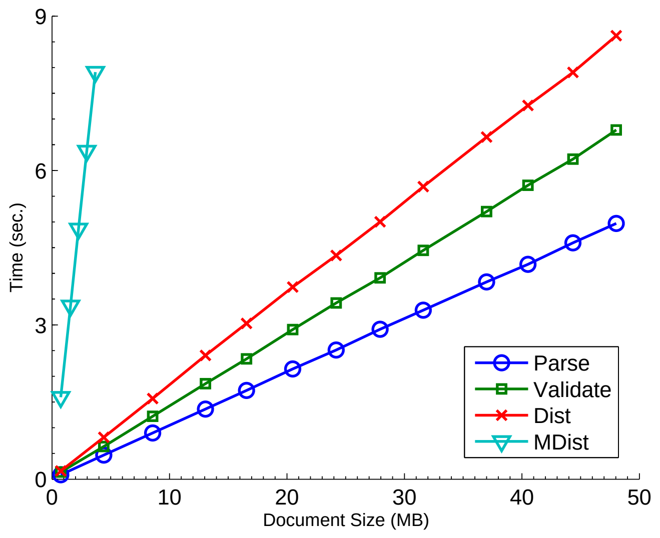


Figure 6.10: Trace graph construction for variable document size (1% invalidity ratio)

should also be applicable to efficiently construct trace graphs. By increasing the size of DTD we also expand Σ whose size is a factor in the time of constructing trace graphs with label modification. This explains why the observed execution time of MDist is cubic in the size of $|D|$.

6.6.2 Valid query answer computation

In the second part of the experiments we measured the time needed to compute the valid query answers. The algorithm QA for computing standard query answers (optimized version of Algorithm 6.1) is used as the baseline. In the experiments we tested performance of two algorithms MVQA and VQA computing valid query answers resp. with and without label modification.

Figure 6.12 shows that for the DTD D_0 computing valid query answers (without modifying operations) is about 6 times longer than computing query answers with QA. Incorporating the modification operation into the model causes again a very significant evaluation time increase.

Because the computation of valid answers involves constructing trace graphs, similarly to computing edit distance, we observe in Figure 6.13 a quadratic dependence between the

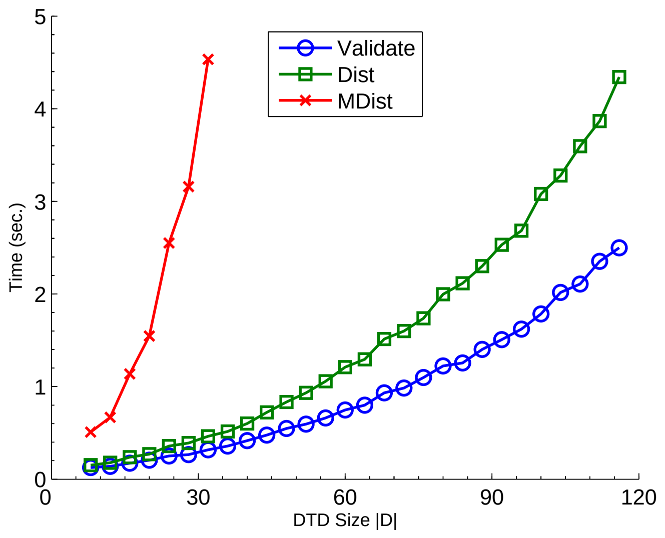


Figure 6.11: Trace graph construction for variable DTD size (1MB Document with 1% invalidity ratio)

performance time and the size of DTD for VQA (because of significantly higher readings of MVQA we omit them).

6.7 Related work

6.7.1 Other editing operations

By extending the repertoire of considered editing operations, our basic framework of repairs could be adapted to handle other common causes of validity violations. We note, however, those extensions may require new algorithms to compute valid answers.

Missing or superfluous inner nodes. The validity of a document that is missing (or has a superfluous) an inner node can be restored by performing a general version of *insertion (resp.* deletion*)* operation. A general insertion of a node takes a subsequence of subelements and links them as the children of the inserted node. The general deletion works in the opposite way: the children of the removed node are linked to the parent of the node. This *generalized* notion of edit distance [SZ97, Bil03] subsumes ours (our notion is sometimes called *1-degree edit distance* [Sel77]). This notion has practical applications in information

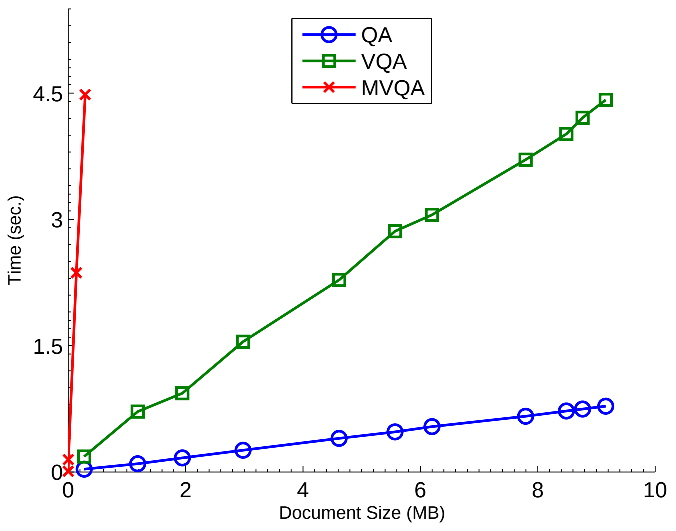


Figure 6.12: Valid query answers computation for variable document size (1% invalidity ratio)

extraction [dCRGdSL04] and integration of XML repositories [GJK$^+$02].

[Suz05] presents a polynomial algorithm for computing the generalized edit distance between a document T and a DTD. This algorithm creates graph structures which, analogously to trace graphs, are used to find optimal repairing sequences by selecting the shortest paths. The construction of those structures takes $O(|T|^5)$ time. In our preliminary research we were able to generalize the tract graph to handle general editing operations in time $O(|D|^4 \times |T|^4)$. It is yet to be seen if the generalized version of the trace graph can be used to compute valid query answers.

Element transpositions Validity violations caused by incorrect order of elements could be addressed by an operation *moving* a subtree to a specified location [Bil03]. This kind of an operation is studied (together with insertion, deletion, and modification) in the context of detecting document changes [CRGMW96, CAM02] and document correction techniques [BdR04]. It is an open question if our framework can be extended to include this operation, however the intractability of the problem of calculating the string edit distance with moves [BdR04, CM02] is discouraging.

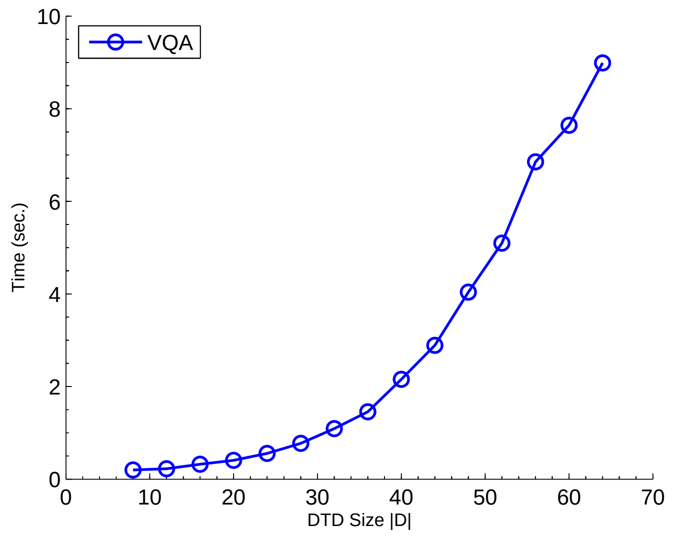


Figure 6.13: Valid query answers computation for variable DTD size (1 MB document with 1% invalidity ratio)

6.7.2 Query approximation

A framework for evaluating queries with none or only a partial knowledge of the schema is proposed in [LYJ04]. A (refined) notion of Lowest Common Ancestor is used to identify meaningful relationships between document elements in situations where expected document structure is not known. In this approach, however, the schema is not used directly in the query evaluation but rather the user is required to assess her knowledge of the schema during query formulation and identify potential structural ambiguity. This contrasts with our approach, where the user is assumed to possess a good knowledge of the expected document structure but cannot or does not wish to identify all potential structural discrepancies. Also, the schema is used during the query evaluation to alleviate the impact of possible validly violations on the query answer.

6.7.3 Structural restoration

The problem of correcting a *slightly invalid* document is considered in [BdR04]. Under certain conditions, the proposed algorithm returns a valid document whose distance from the original one is guaranteed to be within a multiplicative constant of the minimum distance.

The setting is different from ours: XML documents are encoded as binary trees, insertion occurs on an edge and introduces two new nodes and two new edges, and deletion inverts the effect of an insertion (performing those operations on a encoded XML document may shift nodes between levels).

A notion equivalent to the distance of a document to a DTD (Definition 6.4) was used to construct error-correcting parsers for context-free languages [AP72].

6.7.4 Consistent query answers for XML

[FFGZ03] investigates querying XML documents that are valid but violate functional dependencies. Two repairing actions are considered: updating element values with a *null* value and marking nodes as unreliable. This choice of actions prevents from introducing further validity violations in the document upon repairing it. Nodes with null values or marked as unreliable do not cause violations of functional dependencies but also are not returned in the answers to queries. Repairs are consistent instances with a minimal set of nodes affected by the repairing actions. Two types of query semantics are proposed: *certain answers* – answers present in every repair; and *possible answers* – answers present in any repair. A polynomial algorithm for computing certain and possible answers is presented. We note that only simple descending path queries $a_1/a_2/\ldots/a_n$ are considered. This work contains no experimental evaluation of the proposed framework.

A set of operations similar to ours is considered for consistent querying of XML documents that violate functional dependencies in [FFGZ05]. Depending on the operations used different notions of repairs are considered: *cleaning* repairs obtained only by deleting elements, *completing* repairs obtained by inserting nodes, and *general* repairs obtained by both operations. A set-theoretic notion of minimality is used when defining repairs: the set of operations needed to repair a document is minimized with the preference for inserting operations nodes (the document $T_1 = \mathtt{C}(\mathtt{A}(\mathtt{d}), \mathtt{B}(\mathtt{e}), \mathtt{B})$ has only one repair $\mathtt{C}(\mathtt{A}(\mathtt{d}), \mathtt{B}, \mathtt{A}, \mathtt{B})$. Again possible and certain answers are considered. For restricted classes of functional dependencies and DTD's a polynomial algorithm is proposed to compute certain answers to conjunctions of path expressions (basically n-ary $\mathcal{XC}$ queries)

Also this work contains no experimental evaluation of the proposed framework.

[Vll03] is another adaptation of consistent query answers to XML databases closely based on the framework of [ABC99].

Chapter 7

Conclusions and future work

7.1 Universal constraints

In Chapter 3 we investigated the complexity of computing consistent query answers in the presence of universal constraints. We proposed an extended version of a conflict hypergraph that allows to capture conflicts created not only by the *presence* of some tuples but also by the simultaneous *absence* of other tuples. We also showed that an extended conflict graph is a compact representation of all repairs: essentially every repair is a $<_I$-minimal independent set of the extended conflict graph. This property is essential for using extended conflict graphs to compute consistent query answers.

Extending the notions of conflicts to include negative facts leads, however, to a significant increase of computational complexity: computing consistent answers to atomic queries in the presence of universal constraints is Π^2_P-complete (in terms of data complexity). The problem becomes coNP-complete when we restrict the conflicts to involve at most one missing tuple, which corresponds to using constraints with at most one positive atom. If we further restrict the integrity constraints to acyclic sets, then the problem of consistent answering becomes tractable for quantifier-free queries. Consequently, we present an extension of the algorithm of Chomicki and Marcinkowski [CM05] that finds if *true* is the consistent answer to a closed quantifier-free FOL query.

The summary of computational complexity results is presented in Table 7.1; its last row is taken from [CM05].

Constraints	Repair Checking	Consistent Answers to	
		$\{\forall, \exists\}$-free queries	conjunctive queries
Universal	coNP-complete	Π_p^2-complete	
Full TGDs	PTIME	coNP-complete	
Acyclic full TGDs	PTIME	PTIME	coNP-complete
Denial	PTIME	PTIME	coNP-complete

Table 7.1: Summary of complexity results for universal constraints.

We envision several possible directions of future study. First, we note that in our definition of universal constraints we restricted the set of the variables used in positive atoms to be contained in the set of variables used in negative atoms. This a commonly accepted *safety* requirement [AHV95] as constraints not satisfying this requirement may have no database instance satisfying them. For instance, there is no database satisfying the constraint $\forall x.P(x)$. It is interesting to find if the restriction is removed, we observe an increase of complexity of consistent query answering.

Another interesting challenge is generalization of the polynomial algorithm (Theorem 3.20) to handle sets of constraints that are not acyclic or have more than one positive atom. Because of the negative complexity results, we cannot expect that a generalized algorithm would work in polynomial time (unless $P = NP$). We believe, however, that in most practical cases the algorithm should not require exponential time. Consequently, the algorithm could be extended to handle FOL queries with quantifiers. We also note that in this case we cannot guarantee that the algorithm would work in polynomial time as the problem of consistent answering to arbitrary FOL queries is coNP-complete in the presence of denial constraints [CM05].

7.2 The system Hippo

In Chapter 4 we used the positive results from Chapter 3 to build an effective system for computation of consistent answers to projection-free SQL queries. We assumed that the number of conflicts is small enough to allow the active part of the extended conflict hypergraph be stored in main memory. We also proposed a number of optimizations techniques applicable in the context of computing consistent query answers.

We performed a perliminary experimental evaluation of the Hippo system. The results show that the system is very scalable w.r.t. the size of the database and the total number of conflicts. The experiments also show that consistent query answering using conflict hypergraphs is considerably faster than query rewriting.

The main shortcoming of the Hippo system lies in the limited class of queries and integrity constraints for which consistent answers can be computed: only projection-free SQL queries, full TGDs, and denial constraints are handled. Computing consistent answers to queries with projection is known to be intractable in general. However, it would be interesting to see if the rewriting techniques of Fuxman and Miller [FM05] could be adapted to conflict hypergraphs and if combining those two approaches would minimize the overhead inherent to rewriting techniques.

We also believe that the factorization techniques of INFOMIX [EFGL07] can be effectively adapted to conflict graphs to compute consistent answer to queries with projections in the presence of general universal constraints.

7.3 Preferences

In Chapter 5 we proposed a general framework of preferred repairs and preferred consistent query answers. We also proposed a set of desired properties that a family of preferred repairs should satisfy. We observe that introducing a non-trivial input, a priority, into the framework of consistent query answers comes with the price of a significant increase in the computational complexity. Computing $\mathcal{G}$-preferred consistent query answers is Π_p^2-complete. To alleviate this situation we propose two computationally more attractive, *"approximations"*: $\mathcal{P}$- and $\mathcal{C}$-preferred consistent query answers. However, our results also show that non-trivial computation of consistent query answers leads to intractability (Theorem 5.17). Finally, we shown that for one FD per relation preferred consistent query answering is in PTIME for all three presented families of preferred repairs.

The summary of computational complexity results is presented in Table 7.2; its first row is taken from [CM05].

We envision several directions for further work. We plan to investigate other interesting

	Repair Checking	Consistent Answers to	
		$\{\forall, \exists\}$-free queries	conjunctive queries
Rep	PTIME	PTIME	coNP-complete
$\mathcal{G}Rep$	coNP-complete	Π^2_p-complete	
$\mathcal{P}Rep$	PTIME	coNP-complete	
$\mathcal{C}Rep$	PTIME	coNP-complete	

Table 7.2: Summary of complexity results for preferences.

ways of selecting preferred repairs with priorities. Also, extending our approach to cyclic priorities is an interesting and challenging issue. Including priorities in similar frameworks of preferences [GL04] leads to losing $\mathcal{P}2$ (monotonicity). A modified, conditional, version of $\mathcal{P}2$ monotonicity may be necessary to capture non-trivial families of repairs.

Along the lines of [ABC$^+$03b], the computational complexity results could be further studied by assuming the conformance of functional dependencies with BCNF.

Finally, one can extend our framework to handle a broader class of constraints. Conflict graphs can be generalized to hypergraphs [CM05] that are necessary to deal with denial constraints. Then, more than two tuples can be involved in a single conflict and the current notion of priority does not have a clear meaning.

7.4 XML databases

In Chapter 6 we have investigated the problem of querying XML documents containing violations of validity of a local nature, caused by missing or superfluous (leaf) nodes or incorrect node labeling. We proposed a framework that considers possible repairs of a given document, obtained by applying a minimum number of operations that insert, delete, or rename nodes. We studied the complexity of computing the valid answers to a query, and we showed that the problem is in PTIME if we consider only descending paths, i.e. no union, sliding and ascending axes, or no join conditions. We also showed that if we relax any of the restrictions, computing valid query answers becomes intractable. This contrasts with the complexity of query answers, which is known to be in PTIME [GKP02].

We envision several possible directions for future work. First, one can investigate if valid answers can be obtained using query rewriting [GT03, GT04]. Techniques based on query rewriting have been successfully used to compute consistent query answers in relational databases [ABC99, FFM05].

Second, the data complexity of valid query answering, i.e. complexity measured only in terms of the size of the input document, is an important open question. We observe that the proof of coNP-completeness of valid query answering for $\mathcal{XC}(=)$ (Theorem 6.23) uses a query that is not dependent on the input 3CNF formula. Hence we also proved that the data-complexity of valid query answering for $\mathcal{XC}(=)$ is coNP-complete. When publishing preliminary results [SC06], we were strongly convinced that using *disjunctive* implications of tree facts (whose number is exponential in the size of the query) allows us to compute valid query answers in time polynomial in the size of the input document. We encountered, however, technical difficulties proving completeness of our algorithm. Because we have not found any counterexample, we plan to investigate this problem further.

Finally, it would be interesting to find out to what extent our framework can be adapted to handle semantic inconsistencies in XML documents, for example, violations of key dependencies.

Bibliography

[ABC99] M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 68–79, 1999.

[ABC03a] M. Arenas, L. Bertossi, and J. Chomicki. Answer Sets for Consistent Query Answering in Inconsistent Databases. *Theory and Practice of Logic Programming*, 3(4-5):393–424, 2003.

[ABC$^+$03b] M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan, and J. Spinrad. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science (TCS)*, 296(3):405–434, 2003.

[AFM06] P. Andritsos, A. Fuxman, and R. J. Miller. Clean Answers over Dirty Databases: A Probabilistic Approach. In *International Conference on Data Engineering (ICDE)*, page 30, 2006.

[AHV95] S. Abiteboul, R. Hull, and V Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[AMR$^+$98] S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, and J. L. Wiener. Incremental Maintenance for Materialized Views over Semistructured Data. In *International Conference on Very Large Data Bases (VLDB)*, pages 38–49, 1998.

[AP72] A. V. Aho and T. G. Peterson. A Minimum Distance Error-Correcting Parser for Context-Free Languages. *SIAM Journal on Computing*, 1(4):305–312, 1972.

[Bar03] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[BB03] P. Barcelo and L. Bertossi. Logic Programs for Querying Inconsistent Databases. In *International Symposium on Practical Aspects of Declarative Languages (PADL)*, volume 2562 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2003.

[BBFL05] L. Bertossi, L. Bravo, E. Franconi, and A. Lopatenko. Complexity and Approximation of Fixing Numerical Attributes in Databases Under Integrity Constraints. In *International Symposium on Database Programming Languages*, pages 262–278, 2005.

[BC03] L. Bertossi and J. Chomicki. Query Answering in Inconsistent Databases. In Chomicki et al. [CvdMS03], pages 43–83.

[BdR04] U. Boobna and M. de Rougemont. Correctors for XML Data. In *International XML Database Symposium (Xsym)*, pages 97–111, 2004.

[Ber06] L. Bertossi. Consistent Query Answering in Databases. *SIGMOD Record*, 35(2):68–76, June 2006.

[BFFR05] P. Bohannon, M. Flaster, W. Fan, and R. Rastogi. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *ACM SIGMOD International Conference on Management of Data*, pages 143–154, 2005.

[BFG05] M. Benedikt, W. Fan, and F. Geerts. XPath Satisfiability in the Presence of DTDs. In *ACM Symposium on Principles of Database Systems (PODS)*, 2005.

[Bil03] P. Bille. Tree Edit Distance, Aligment and Inclusion. Technical Report TR-2003-23, The IT University of Copenhagen, 2003.

[BPV04] A Balmin, Y. Papakonstantinou, and V. Vianu. Incremental Validation of XML Documents. *ACM Transactions on Database Systems (TODS)*, 29(4):710–751, December 2004.

[Bre89] G. Brewka. Preferred Subtheories: An Extended Logical Framework for Default Reasoning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1043 – 1048, 1989.

[CAM02] G. Cobena, S. Abiteboul, and A. Marian. Detecting Changes in XML Documents. In *International Conference on Data Engineering (ICDE)*, pages 41–52, 2002.

[Cho03] J. Chomicki. Preference Formulas in Relational Queries. *ACM Transactions on Database Systems (TODS)*, 28(4):427–466, December 2003.

[Cho07] J. Chomicki. Consistent Query Answering: Five Easy Pieces. In *International Conference on Database Theory (ICDT)*, pages 1–17, 2007.

[CLR03] A Cali, D. Lembo, and R. Rosati. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 260–271, 2003.

[CM02] G. Cormode and S. Muthukrishnan. The String Edit Distance Matching Problem with Moves. In *ACM SIAM Symposium on Discrete Algorithms (SODA)*, pages 667–676, 2002.

[CM05] J. Chomicki and J. Marcinkowski. Minimal-Change Integrity Maintenance Using Tuple Deletions. *Information and Computation*, 197(1-2):90–121, February 2005.

[CRGMW96] S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change Detection in Hierarchically Structured Information. In *ACM SIGMOD International Conference on Management of Data*, pages 493–504, 1996.

[CvdMS03] J. Chomicki, R. van der Meyden, and G. Saake, editors. *Logics for Emerging Applications of Databases*. Springer-Verlag, 2003.

[dCRGdSL04] D. de Castro Reis, P. Golgher, A. da Silva, and A Laender. Automatic Web News Extraction using Tree Edit Distance. In *International Conference on World Wide Web (WWW)*, pages 502–511, 2004.

[DJ03] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley, 2003.

[EBR$^+$01] S. M. Embury, S. M. Brandt, I. Robinson, J. S.and Sutherland, F. A. Bisby, W. A. Gray, A. C. Jones, and R. J. White. Adapting Integrity Enforcement Techniques for Data Reconciliation. *Information Systems*, 26(8):657–689, 2001.

[EFGL03] T. Eiter, M. Fink, G. Greco, and D. Lembo. Efficient Evaluation of Logic Programs for Querying Data Integration Systems. In *International Conference on Logic Programming (ICLP)*, pages 163–177, 2003.

[EFGL07] T. Eiter, M. Fink, G. Greco, and D. Lembo. Repair Localization for Query Answering from Inconsistent Databases. Technical Report 1843-07-01, Institut Für Informationssysteme, Technische Universität Wien, 2007.

[EFLP00] T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative Problem-Solving in DLV. In J. Minker, editor, *Logic-Based Artificial Intelligence*, pages 79–103. Kluwer Academic Publishers, 2000.

[FFGZ03] S. Flesca, F. Furfaro, S. Greco, and E. Zumpano. Repairs and Consistent Answers for XML Data with Functional Dependencies. In *International XML Database Symposium (Xsym)*, volume 2824 of *Lecture Notes in Computer Science*, pages 238–253. Springer, 2003.

[FFGZ05] S. Flesca, F Furfaro, S. Greco, and E. Zumpano. Querying and Repairing Inconsistent XML Data. In *Web Information Systems Engineering (WISE)*, pages 175–188, 2005.

[FFM05] A. Fuxman, E. Fazli, and R. J. Miller. ConQuer: Efficient Management of Inconsistent Databases. In *ACM SIGMOD International Conference on Management of Data*, pages 155–166, 2005.

[FGZ04] S. Flesca, S. Greco, and E. Zumpano. Active Integrity Constraints. In *ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 98–107, 2004.

[FM05] A. Fuxman and R. J. Miller. First-Order Query Rewriting for Inconsistent Databases. In *International Conference on Database Theory (ICDT)*. Springer, 2005.

[FUV83] R. Fagin, J. D. Ullman, and M. Y. Vardi. On the Semantics of Updates in Databases. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 352–356, 1983.

[Fux07] A. Fuxman. *Efficient Query Processing Over Inconsistent Databases*. PhD thesis, University of Toronto, 2007.

[GGZ01] G. Greco, S. Greco, and E. Zumpano. A Logic Programming Approach to the Integration, Repairing and Querying of Inconsistent Databases. In *International Conference on Logic Programming (ICLP)*, volume 2237 of *Lecture Notes in Computer Science*, pages 348–364. Springer, 2001.

[GGZ03] G. Greco, S. Greco, and E. Zumpano. A Logical Framework for Querying and Repairing Inconsistent Databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1389–1408, 2003.

[GJK$^+$02] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. Approximate XML Joins. In *ACM SIGMOD International Conference on Management of Data*, pages 287–298, 2002.

[GKP02] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. In *International Conference on Very Large Data Bases (VLDB)*, pages 95–106, 2002.

[GL04] G. Greco and D. Lembo. Data Integration with Preferences Among Sources. In *International Conference on Conceptual Modeling (ER)*, pages 231–244. Springer, November 2004.

[Gro97] B. N. Grosof. Prioritized Conflict Handling for Logic Programs. In *International Logic Programming Symposium*, pages 197–211, 1997.

[GSTZ04] S. Greco, C. Sirangelo, I. Trubitsyna, and E. Zumpano. Feasibility Conditions and Preference Criteria in Quering and Repairing Inconsistent Databases. In *International Conference on Database and Expert Systems Applications (DEXA)*, pages 44–55, 2004.

[GT03] G. Grahne and A. Thomo. Query Containment and Rewriting using Views for Regular Path Queries under Constraints. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 111–122, 2003.

[GT04] G. Grahne and A. Thomo. Query Answering and Containment for Regular Path Queries under Distortions. In *Foundations of Information and Knowledge Systems (FOIKS)*, pages 98–115, 2004.

[Hal97] J. Y. Halpern. Defining Relative Likehood in Partially-Ordered Preferential Structures. *Journal of Artificial Intelligence Research*, 1997.

[HMU01] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 2nd edition, 2001.

[KSS03] A. Klarlund, T. Schwentick, and D. Suciu. XML: Model, Schemas, Types, Logics and Queries. In Chomicki et al. [CvdMS03].

[LB07] A. Lopatenko and L. Bertossi. Complexity of Consistent Query Answering in Databases Under Cardinality-Based and Incremental Repair Semantics. In *International Conference on Database Theory (ICDT)*, pages 179–193, 2007.

[Lib06] L. Libkin. Data Exchange and Incomplete Information. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 60–69, 2006.

[Lom00] David B. Lomet. Letter from the Editor-in-Chief. *IEEE Data Eng. Bull.*, 23(4), 2000.

[Lop06] A. Lopatenko. *Logic Based Data Integration.* PhD thesis, University of Manchester, 2006.

[LRR06] D. Lembo, R. Rosati, and M. Ruzzi. On the First-Order Reducibility of Unions of Conjunctive Queries over Inconsistent Databases. In *EDBT Workshops (IIDB)*, pages 358–374, 2006.

[LYJ04] Y. Li, C. Yu, and H. V. Jagadish. Schema-Free XQuery. In *International Conference on Very Large Data Bases (VLDB)*, pages 72–83, 2004.

[MAA04] A. Motro, P. Anokhin, and A. C. Acar. Utility-based Resolution of Data Inconsistencies. In *International Workshop on Information Quality in Information Systems (IQIS)*, pages 35–43. ACM, 2004.

[Mar04] M. Marx. Conditional XPath, the First Order Complete XPath Dialect. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 13–22, 2004.

[Nev02] F. Neven. Automata, Logic, and XML. In *Workshop on Computer Science Logic (CSL)*, volume 2471 of *Lecture Notes in Computer Science*, pages 2–26. Springer, 2002.

[Pap94] C. Papadimitriou. *Computational Complexity.* Addison Wesley Lengman, 1994.

[RG00] R. Ramakrishnan and J. Gehrke. *Database Management Systems.* WCB/McGraw-Hill, 2000.

[SC06] S. Staworko and J. Chomicki. Validity-Sensitive Querying of XML Databases. In *EDBT Workshops (dataX)*, pages 164–177. Springer, 2006.

[Sel77] S. M. Selkow. The Tree-to-Tree Editing Problem. *Information Processing Letters*, 6(6):184–186, 1977.

[SI00] C. Sakama and K. Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence*, 123:185–222, 2000.

[Suz05] N. Suzuki. Finding an Optimum Edit Script between an XML Document and a DTD. In *ACM Symposium on Applied Computing (SAC)*, pages 647–653, 2005.

[SZ97] D. Shasha and K. Zhang. Approximate Tree Pattern Matching. In A. Apostolico and Z. Galil, editors, *Pattern Matching in Strings, Trees, and Arrays*, pages 341–371. Oxford University Press, 1997.

[Ull97] J. Ullman. Information Integration Using Logical Views. In *International Conference on Database Theory (ICDT)*, pages 19–40, 1997.

[Var82] M. Y. Vardi. The Complexity of Relational Query Languages. In *ACM Symposium on Theory of Computing (STOC)*, pages 137–146, 1982.

[Vll03] S. Vllalobos. Consistent Answers to Queries Posed to Inconsistent XML Databases. Master's thesis, Catholic University of Chile (PUC), 2003. In Spanish.

[VNV02] D. Van Nieuwenborgh and D. Vermeir. Preferred Answer Sets for Ordered Logic Programs. In *European Conference on Logics for Artificial Intelligence (JELIA)*, pages 432–443. Springer-Verlag, LNCS 2424, 2002.

[W3C99] W3C. XML path language (XPath 1.0), 1999. `http://www.w3.org/TR/xpath`.

[Wij03] J. Wijsen. Condensed Representation of Database Repairs for Consistent Query Answering. In *International Conference on Database Theory (ICDT)*, volume 2572 of *Lecture Notes in Computer Science*, pages 378–393. Springer, 2003.

[Wij05] J. Wijsen. Database Repairing Using Updates. *ACM Transactions on Database Systems (TODS)*, 30(3):722–768, 2005.

Appendix A

Omitted proofs

Propositions 5.8, 5.16, and 5.21 *$\mathcal{C}Rep \sqsubseteq \mathcal{G}Rep \sqsubseteq \mathcal{P}Rep$ and $\mathcal{C}Rep$, $\mathcal{G}Rep$, and $\mathcal{P}Rep$ satisfy $\mathcal{P}1$-$\mathcal{P}4$.*

Proof $\boxed{\mathcal{G}Rep \sqsubseteq \mathcal{P}Rep}$ Trivial from the definitions of global and Pareto optimality.

$\boxed{\mathcal{P}1 \text{ for } \mathcal{G}Rep}$ Assume $\mathcal{G}Rep(I, F, \succ)$ is empty. This is possible only if all $\gg$-chains are infinite (no $\gg$-maximal element). Because there is only a finite number of repairs, $\gg$ is cyclic. This, however, implies that $\succ$ is cyclic as well; a contradiction.

$\boxed{\mathcal{P}1 \text{ for } \mathcal{P}Rep}$ by $\mathcal{P}1$ for $\mathcal{G}Rep$ and $\mathcal{G}Rep \sqsubseteq \mathcal{P}Rep$.

$\boxed{\mathcal{P}2 \text{ for } \mathcal{G}Rep}$ Take any two acyclic priorities $\succ_1$ and $\succ_2$ such that $\succ_1 \subseteq \succ_2$ and any $I' \in \mathcal{G}Rep(I, F, \succ_2)$. Suppose that I' is not globally optimal w.r.t. $\succ_1$, i.e. there exists a nonempty $X \subseteq I'$ and a nonempty $Y \subseteq I$ s.t.

$$\forall x \in X. \exists y \in Y. y \succ_1 x.$$

Because $\succ_1 \subseteq \succ_2$ then also

$$\forall x \in X. \exists y \in Y. y \succ_2 x,$$

which contradicts global optimality of I' w.r.t. $\succ_1$.

$\boxed{\mathcal{P}2 \text{ for } \mathcal{P}Rep}$ Proved analogously to $\mathcal{P}2$ for $\mathcal{G}Rep$.

$\boxed{\mathcal{C}Rep \sqsubseteq \mathcal{G}Rep}$ Trivial by $\mathcal{P}1$ and $\mathcal{P}2$ for $\mathcal{G}Rep$.

$\boxed{\mathcal{P}1 \text{ for } \mathcal{C}Rep}$[1] In Algorithm $\mathcal{PCR}$, for an acyclic priority $\succ$ the set $\omega_{\succ}(s)$ is empty if

[1] We note that Theorem 5.20 used here uses $\mathcal{P}4$ for $\mathcal{G}Rep$ and $\mathcal{P}Rep$, and $\mathcal{G}Rep \sqsubseteq \mathcal{P}Rep$. As it can be,

and only if s is empty. In every iteration an element x is selected and removed from s. Since s is initialized with a finite set of tuples I, the algorithm terminates and returns a finite set of elements I'.

$\boxed{\mathcal{P}2 \text{ for } \mathcal{C}Rep}$ Trivial from the definition of $\mathcal{C}Rep$.

$\boxed{\mathcal{P}3 \text{ for } \mathcal{C}Rep}$[1] Because $\omega_{\varnothing}(I) = I$, Algorithm $\mathcal{PCR}$ with an empty priority constructs a maximal independent set of the conflict graph. Moreover, it is also easy to see that every maximal independent set can be a result of Algorithm $\mathcal{PCR}$.

$\boxed{\mathcal{P}3 \text{ for } \mathcal{G}Rep}$ Trivially by $\mathcal{C}Rep \sqsubseteq \mathcal{G}Rep$ and $\mathcal{P}3$ for $\mathcal{C}Rep$.

$\boxed{\mathcal{P}3 \text{ for } \mathcal{P}Rep}$ Also, trivially by $\mathcal{C}Rep \sqsubseteq \mathcal{P}Rep$ and $\mathcal{P}3$ for $\mathcal{C}Rep$.

$\boxed{\mathcal{P}4 \text{ for } \mathcal{P}Rep}$ Take any total and acyclic priority $\succ$ and assume that there exist two different Pareto optimal repairs I_1 and I_2. Naturally, both $I_1 \setminus I_2$ and $I_2 \setminus I_1$ are nonempty. From Pareto optimality for I_1 we have that

$$\forall x \in I_1 \setminus I_2 . \exists y \in I_2 \setminus I_1 . x \not\succ y.$$

Because $\succ$ is total this is equivalent to

$$\forall x \in I_1 \setminus I_2 . \exists y \in I_2 \setminus I_1 . y \succ x.$$

Similarly from Pareto optimality fr I_2 we have

$$\forall y \in I_2 \setminus I_1 . \exists y \in I_1 \setminus I_2 . x \succ y.$$

Now, we construct an infinite sequence $y_0, x_1, y_1, \ldots$ as follows:

- y_0 is any element of $I_2 \setminus I_1$,
- x_i is any element of $I_1 \setminus I_2$ such that $x_i \succ y_{i-1}$,
- y_i is any element of $I_2 \setminus I_1$ such that $y_i \succ x_i$.

Note that $y_0 \succ x_1 \succ y_1 \succ x_2 \succ y_2 \ldots$ and because I_1 and I_2 are finite, there must be repetitions in the sequence and consequently a cycle in $\succ$; a contradiction.

however, easily checked, there is no cyclic dependency in our proofs.

$\boxed{\mathcal{P}4 \text{ for } \mathcal{G}Rep}$ Trivially from $\mathcal{P}4$ for $\mathcal{P}Rep$, $\mathcal{P}1$ for $\mathcal{G}Rep$, and $\mathcal{G}Rep \sqsubseteq \mathcal{P}Rep$.

$\boxed{\mathcal{P}4 \text{ for } \mathcal{C}Rep}$ Also trivially from $\mathcal{P}4$ for $\mathcal{P}Rep$, $\mathcal{P}1$ for $\mathcal{C}Rep$, and $\mathcal{C}Rep \sqsubseteq \mathcal{P}Rep$. □

Lemma 5.26 *A globally optimal (common) repair I' satisfying $\neg\Phi_i$ exists if and only if the following conditions are satisfied:*

1. $\{t_1, \ldots, t_k\}$ *is conflict-free;*

2. $\{D_{t_1}, \ldots, D_{t_k}\} \cap \{D_{t_{k+1}}, \ldots, D_{t_m}\} = \varnothing$*;*

3. $D_{t_j} \cap \omega_{\succ}(C_{t_j}) \neq \varnothing$ *for every* $j \in \{1, \ldots, k\}$.

4. $\omega_{\succ}(C_{t_j}) \setminus (D_{t_{k+1}} \cup \ldots \cup D_{t_n}) \neq \varnothing$ *for every* $j \in \{k+1, \ldots, n\}$.

Proof $\boxed{\Rightarrow}$ We take any globally optimal repair I' satisfying $\neg\Phi_i$. 1 and 2 are trivially satisfied.

Assume that I' is the result of Algorithm $\mathcal{PCR}$ with the sequence of choices made in Step 5.20 $s_1, \ldots, s_l$. Take any $j \in \{1, \ldots, k\}$ and let j' be the smallest index of a tuple from C_{t_j} in the sequence. Because t_j is present in the repair I', $s_{j'} \in D_{t_j}$. Also, prior to making the choice $s_{j'}$ the temporary instance J contains C_{t_j} and $s_{j'} \in \omega_{\succ}(J)$. This implies $s_{j'} \in \omega_{\succ}(C_{t_j})$ which proves 3.

We show 4 similarly. First, we recall that $\omega_{\succ}$ is monotonic and observe that for any $j \in \{k+1, \ldots, n\}$ the cluster C_{t_j} can have elements in common only with (X, Y)-cluster of those tuples $t_{k+1}, \ldots, t_n$ that belong to C_{t_j}. For any $j \in \{k+1, \ldots, n\}$ let j' be the smallest index of a tuple from the sequence of choices used to construct I'. Prior to making the choice $s_{j'}$ the temporary instance J contains C_{t_j}, $s_{j'} \in \omega_{\succ}(C_{t_j})$, and s_j does not belong to any of $D_{t_{k+1}}, \ldots, D_n$.

$\boxed{\Leftarrow}$ We construct I' using Algorithm $\mathcal{PCR}$ by specifying the sequence $s_1, \ldots, s_l$ of choices made in Step 5.20. For $j \in \{1, \ldots, k\}$ the choice s_j is any tuple from $D_{t_j} \cap \omega_{\succ}(C_{t_j})$ (possible by 1 and 3). For any $j \in \{k+1, \ldots, n\}$ the choice s_j is any tuple from $\omega_{\succ}(C_{t_j}) \setminus (D_{t_{k+1}} \cup \ldots \cup D_{t_n})$ (possible by 2 and 4). The remaining choices are selected by Algorithm in an arbitrary way. Finally, we observe that the first k steps guarantees that the tuples $t_1, \ldots, t_k$ belong to the repair instance I' (possibly placed there in later consecutive steps) and that

I' does not contain any of the tuples $t_{k+1}, \ldots, t_n$. □

Lemma 5.27 *A Pareto optimal repair I' satisfying $\neg\Phi_i$ exists if and only if the following conditions are satisfied:*

1. *$\{t_1, \ldots, t_k\}$ is conflict-free;*
2. *$\{D_{t_1}, \ldots, D_{t_k}\} \cap \{D_{t_{k+1}}, \ldots, D_{t_m}\} = \varnothing$;*
3. *for every $j \in \{1, \ldots, k\}$, for every tuple $t \in C_{t_j} \setminus D_{t_j}$ there exists $t' \in D_{t_j}$ such that $t \not\succ t'$.*
4. *for every $j \in \{k+1, \ldots, n\}$ there exists an (X,Y)-cluster D of C_{t_j} different from $D_{t_{k+1}}, \ldots, D_{t_n}$ such that for every $t \in D_{t_{k+1}} \cup \ldots \cup D_{t_n}$, there exists $t' \in D$ such that $t \not\succ t'$.*

Proof $\boxed{\Leftarrow}$ We construct the repair I' by selecting an (X,Y)-cluster from every X-cluster. Because Pareto optimality is defined in terms of conflicting tuples and for one FD conflicts can be present only inside an X-cluster, to show that the repair I' is Pareto optimal it is enough to show the for every X-cluster the selected (X,Y)-cluster is Pareto optimal (among all (X,Y)-clusters in the X-cluster).

For X-clusters $C_{t_1}, \ldots, C_{t_k}$ we select $D_{t_1}, \ldots, D_{t_k}$ resp. We note that by 1 the (X,Y)-clusters belong to different X-clusters and by 2 we do not include any of the tuples $t_{k+1}, \ldots, t_m$. Pareto optimality is implied by 3.

For X-clusters $C_{t_{k+1}}, \ldots, C_{t_m}$ we select the (X,Y)-clusters as described in 4. Pareto optimality of those clusters is also implied by 4.

For an X-cluster other than $C_1, \ldots, C_m$ we select any (X,Y)-cluster that is a Pareto optimal repair of the X-cluster.

Since all selected (X,Y)-clusters are Pareto optimal, the instance I' is a Pareto optimal repair such that $I' \models \neg\Phi_i$.

$\boxed{\Rightarrow}$ 1 and 2 are trivially implied by $I' \models \neg\Phi_i$. To show 3 and 4 we observe that a Pareto optimal repair contains exactly one Pareto optimal (X,Y)-cluster for every X-cluster. For clusters $C_{t_1}, \ldots, C_{t_k}$ this together with the fact that $\{t_1, \ldots, t_k\} \subseteq I'$ implies 3. For clusters $C_{t_{k+1}}, \ldots, C_{t_m}$ this together with the fact that $\{t_{k+1}, \ldots, t_n\} \cap I' = \varnothing$ implies 4. □