Design Principles for XML Data

by

Marcelo Arenas

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Design Principles for XML Data

Marcelo Arenas
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2005

In this dissertation, we take a first step towards the design and normalization theory for XML documents. We start by noticing that while in the relational world the criteria for being well designed are very intuitive, they become more obscure when one moves to XML. Thus, our first contribution is to provide a tool for testing when a condition on a database design, specified as a normal form, corresponds to a good design. We use techniques of information theory, and define a measure of information content of elements in a database with respect to a set of constraints. This measure can be used in different data models, in particular, we use it in the relational model to provide information-theoretic justification for well-known normal forms and for normalization algorithms.

As our second contribution we introduce languages for XML data dependencies, that will be used later as the source of semantic information in the design of XML databases. Since inconsistent XML specifications may arise in practice because of the interaction between these dependencies and the constraint imposed by XML schemas (DTDs), our next contribution is to pinpoint the complexity of checking consistency of XML specifications.

We then show that XML documents may contain redundant information, and may be prone to update anomalies. Thus, our final contribution is to define an XML normal form, XNF, that avoids update anomalies and redundancies. We study its properties, and show that it generalizes BCNF and that it can be justified by our information-theoretic measure. We present an algorithm for converting any XML schema into an equivalent one in XNF, and we use our information-theoretic measure to justify this algorithm.

Para Vanny y Magdalena

Acknowledgements

I would especially like to thank the following people.

- First and foremost, I would like to thank my supervisor, Professor Leonid Libkin, for his extraordinary support in all aspects of my graduate life. I do not know of any other supervisor who spends so much time with his students. To meet with him I just had to drop by his office, and then I could spend hours talking about research and life. His encouragement, enthusiasm and single malts have made working with him a great pleasure.

- A Vanny, quien decidió embarcarse conmigo en esta aventura. Gracias por compartir conmigo este sueño. Te amo!

- A Magdalena, por esperarme todos los días, por hacerme reir tanto y por dejarme dormir poco.

- I would like to thank the members of my thesis committee, Professors Renee Miller, Mariano Consens and Hector Levesque, for reading my thesis and providing so many useful suggestions.

- I would like to thank Professor Alberto Mendelzon, who was part of my thesis committee but unfortunately passed away before this work was finished. Alberto was an exceptional human being and, in particular, because of him I applied for a Ph.D at the University of Toronto and joined the database group.

- I would like to thank the external member of my thesis committee, Professor Moshe Vardi, for taking the time to read my thesis and for providing useful comments. I feel honored to have had him on my thesis committee.

- A mis padres, por todo el apoyo durante todos estos años. No creo que ellos sepan todo lo agradecido que estoy por su dedicación.

- A mi suegra, quien también nos ayudo mucho en todo el proceso. Tampoco creo que ella sepa todo lo agradecido que estoy por esto.

- I would like to thank my good friend Tasos Kementsietsidis. After sharing an office with him for two years, staying awake the whole night writting a paper, drinking some vodka mandarin and visiting him in his new home in Edinburgh, I finally came to the conclusion that Greeks do not know how to play soccer.

- I would like to thank Ramona Truta for being such a good friend and for helping me every time I needed something. But mostly, I would like to thank her for cooking the best sarmale in the world.
- A Pablo Barceló por mantenerme informado de toda la actualidad nacional.
- I would also like to thank my personal trainer Jorge Baier for keeping me in good shape (sure!).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Information is one of the most –if not the most– valuable assets of a company. Therefore, organizations need tools to allow them to structure, query and analyze their data, and, in particular, they need tools providing simple and fast access to their information.

During the last 30 years, relational databases have become the most popular computer application for storing and analyzing information. The simplicity and elegance of the relational model, where information is stored just as tables, has largely contributed to this success.

To use a relational database, a company first has to think of its data as organized in tables. How easy is for a user to understand this organization and use the database depends on the design of these relations. If tables are not carefully selected, users can spend too much time executing simple operations or may not be able to extract the desired information.

Since the beginnings of the relational model, it was clear for the research community that the process of designing a database is a nontrivial and time-consuming task. Even for simple application domains, there are many possible ways of storing the data of interest. Soon the difficulties in designing a database became clear for practitioners, and the design problem was recognized as one of the fundamental problems for the relational technology.

During the 70s and 80s, a lot of effort was put into developing methodologies to aid in the process of deciding how to store data in a relational database. The most prominent approaches developed at that time –which today are an standard part of the relational technology– were the entity-relationship and the normalization approaches. In the former approach, a diagram is used to specify the objects of an application domain and the relationships between them. The *schema* of the relational database, i.e. the set of tables and column names, is then automatically generated from the diagram. In the normalization approach, an already designed relational database is given as input, together with some semantics information provided by a user in the form of relationships between different parts of the database, called *data dependencies*. This semantic information is then used to check whether the design has some desirable properties, and if this is not the case, it can also be used to convert the poor design into an equivalent well-designed database.

The normalization approach was proposed in the early 70s by Codd. In this approach, a *normal form*, defined as a syntactic condition on data dependencies, specifies a property that a well-designed database must satisfy. Normalization as a way of producing good relational database designs is a well-understood topic. In the 70s and 80s, normal forms such as 3NF, BCNF, 4NF, and PJ/NF were introduced to deal with the design of relational databases having different types of data dependencies. These normal forms, together with normalization algorithms for converting a poorly designed database into a well-designed database, can be found today in every database textbook.

With the development of the Web, new data models have started to play a more prominent role. In particular, XML (eXtensible Markup Language) has emerged as the standard data model for storing and interchanging data on the Web. As more companies adopt XML as the primary data model for storing information, the problem of designing XML databases is becoming more relevant. After 30 years of database development, it is clear for both researchers and practitioners that the performance of XML databases depends on their design.

The concepts of database design and normal forms are central in relational database technology. In this dissertation, we extend them to XML databases. Our goal is to find principles for good XML data design, and algorithms to produce such designs. We believe this research is especially relevant nowadays, since a huge amount of data is being put on the Web. Once massive Web databases are created, it is very hard to change their organization; thus, there is a risk of having large amounts of widely accessible, but poorly organized legacy data.

In this dissertation, we extend the normalization approach to XML databases. We believe that in order to have tools for helping users in designing XML databases, both the normalization and the entity-relationship approach have to be developed for XML. In relational databases, designers typically follow the methodology of entity-relationship design. However, the importance of the normalization approach in this data model cannot be ignored. In the relational world, normalization theory established some fundamental properties that a well-designed database should possess, and that became the target for the entity-relationship methodology, as this approach generates relational schemas that are in some normal form, typically 3NF or BCNF. We expect the situation to be similar for the case of XML. In practice, the users will utilize some form of entity-relationship diagram to design an XML database, and then an algorithm will generate a schema in some XML normal form. In some cases, normalization algorithms will be also used to fine-tune the resulting schema. But in the case of XML, we expect the normalization approach to play an even more important role in identifying the properties that the generated schemas should possess. The flexibility of XML and the rather expressive languages used for specifying XML schemas make more difficult to identify good properties for XML documents, and today we can find examples of proposals for XML entity-relationship diagrams that have not been able to clearly identify what are the properties of the generated schemas [WLLD01, EM01a, SMD03, LLD04] and, in particular, what are the properties that they should satisfy. We expect the XML normalization approach to give the right answers for these questions.

Designing a relational database means choosing an appropriate relational schema for the data of interest. A relational schema consists of a set of relations, or tables, and a set of data dependencies over these relations. Designing an XML database is similar: An appropriate *XML schema* has to be chosen, which usually consists of a DTD (Document Type Definition) and a set of data dependencies. However, the structure of XML documents, which are trees as opposed to relations, and the rather expressive constraints imposed by DTDs make the design problem for XML databases quite challenging.

1.1 Contributions

Seeking for principles for good XML data design, and algorithms to produce such designs, this dissertation addresses several problems. More specifically, we start by noticing that while in the relational world the criteria for being well designed are usually very intuitive and clear to state, they become more obscure when one moves to more complex data models such as XML. Then we provide a set of tools for testing when a condition on a database design, specified by a normal form, corresponds to a good design. We use techniques of information theory, and define a measure of information content of elements in a relational database with respect to a set of constraints. Our intention when introducing this information-theoretic measure is to have a robust tool that can be used to study normal forms in more complex data models such as XML. We use this measure to provide information-theoretic justification for familiar relational normal forms such as BCNF, 4NF and PJ/NF, and we also look at information-theoretic criteria for justifying normalization algorithms for relational databases.

Once the set of tools for testing when a normal form corresponds to a good design are developed, we address the problem of designing XML databases. We start by introducing a formal model for XML databases, and then noticing that, as in the case of relational databases, the design of XML databases is guided by the semantic information encoded in data dependencies. Thus, our next step is to introduce several languages for XML data dependencies.

We continue our work by observing that XML databases are prone to a serious design problem: As opposed to relational databases, XML databases can be inconsistent in the sense that there is no way of populating the database and satisfying the constraints imposed by its schema. Since inconsistent XML databases are poorly designed, it is desirable to have algorithms for checking consistency. Thus, our next step is to study the problem of checking consistency for a variety of XML data dependency languages. Unfortunately, our main conclusion in this part of the dissertation is that compile-time verification of consistency is usually infeasible.

After dealing with the consistency of XML databases, we study the elements that we need to introduce a normal form for XML documents. More specifically, at this point we propose a functional dependency language for XML documents, which is the basic component of the XML normal form proposed in this dissertation. Once the main properties of this language have been established, we propose a normal form for XML documents. In the final part of this dissertation, we show that, like relational databases, XML documents may contain redundant information, and may be prone to update anomalies. We define an XML normal form, XNF, that avoids update anomalies and redundancies. We study its properties, and show that the information-theoretic measure mentioned above justifies XNF. We present an algorithm for converting any DTD into an equivalent one in XNF, and we finish this dissertation by looking at information-theoretic criteria for justifying this algorithm.

It is worth mentioning that the results of this dissertation appeared in the following publications: the results of Chapter 3 appeared in [AL03, AL05], the results of Chapter 5 appeared in [AFL02a, AFL02b] and the results of Chapters 6 and 7 appeared in [AL02, AL04].

Chapter 2

Relational and Nested Relational Databases

In this chapter, we present normalization theory for relational and nested relational databases. We decided to include the nested relational model because its hierarchical structure is closely related to the hierarchical structure of XML. In fact, in Chapter 7, we show that one of the nested normal forms introduced in this chapter is closely related to the XML normal form proposed in this dissertation.

This chapter is divided into two sections. In the first section, we consider relational databases, and in the second one we consider nested relational databases.

2.1 Relational Databases

To present normalization theory for relational databases, we have divided this section into three sections. In Section 2.1.1, we present the basic notions in the relational model. In Section 2.1.2, we introduce data dependency theory for relational databases, and we describe in detail the results of this theory that are needed for introducing normalization theory in Section 2.1.3.

2.1.1 Basic Notions

The relational model was introduced by Codd [Cod70] in the '70s and today is the most popular data model. In this simple formalism, a database is viewed as a collection of relations or tables. For instance, a relational database storing information about courses in a university is shown in Figure 2.1. Each row of this table contains the number of a course, its title, the number of one of its sections and the room where this section is held.

Number	*Title*	*Section*	*Room*
CSC 258	Computer Organization	1	LP266
CSC 258	Computer Organization	2	GB258
CSC 258	Computer Organization	3	LM161
CSC 258	Computer Organization	3	GB248
CSC 434	Data Management Systems	1	GB248

Figure 2.1: Relation *Course*.

The relation shown in Figure 2.1 consists of a time-varying part, the data about courses, and a part considered to be time independent, the *schema* of the relation. These two parts are the main components of the relational model. Formally, a *relation schema* is an expression of the form $R[U]$, where R is the name of the relation and $U = \{A_1, \ldots, A_n\}$ is the set of its *attributes*. For each attribute $A \in U$, $Dom(A)$ is used to denote its domain. We assume that all domains are infinite[1]. For example, the schema of the relation shown in Figure 2.1 is $Courses[U]$, where $U = \{Number, Title, Section, Room\}$ and $Dom(Section)$ is the set of natural numbers. A U-tuple t is a function with domain U such that for every $A \in U$, $t(A) \in Dom(A)$. Thus, a tuple is a mapping that associates a value to each attribute of U. An *instance* I of a relation schema $R[U]$ is a set of U-tuples. For example, the instance shown in Figure 2.1 contains four tuples, the first of which is defined as: $t_1(Number) =$ CSC 258, $t_1(Title) =$ Computer Organization, $t_1(Section) = 1$ and $t_1(Room) =$ LP266. If an order for the set of attributes is provided, then we represent tuples by enumerating their values: Tuple t_1 in the previous example is represented as (CSC 258, Computer Organization, 1, LP266). A *database schema* is a set of relation schemas $S = \{R_1[U_1], \ldots, R_n[U_n]\}$. An instance I of schema S assigns to each symbol $R[U] \in S$ a relation $I(R)$ which is a finite set of U-tuples.

Usually, the information contained in a database must satisfy some constraints. For example, in the relation shown in Figure 2.1 we expect that only one title is associated to each course number. By providing the schema of a relation, we specify syntactic constraints, the structure of a relation, but we do not specify the semantic constraints that the instances should satisfy. To remedy this, for each relation schema it is necessary

[1] We defer the discussion on finite domains to Section 2.1.2.

to specify separately a set of semantic restrictions. These restrictions are called *data dependencies* and they are expressed by using suitable languages (see Section 2.1.2). For the sake of simplicity, given a relation schema $R[U]$ and a set of data dependencies Σ over R, $(R[U], \Sigma)$ is also called relation schema.

Querying Relational Databases

The most popular query language for relational databases is SQL. Practically all commercial database systems used SQL as the main data manipulation language. On the theory side, probably the most popular query language is relational algebra. This language has been extensively studied in the database community and, particularly, it plays a central role in the normalization theory of relational databases. Some of the most important concepts in this theory, like information losslessness, are defined in terms of relational algebra operators. In this section, we present the core operators of this algebra, which are expressive enough to capture the most common SQL statements.

Relational algebra has five basic operators [AHV95]: selection, projection, join, union and difference. The first two operators are unary and the remaining ones are binary. These operators are defined as follows. Let $R[U]$ be a relation schema and I an instance of $R[U]$. The two primitive forms of the selection operator are $\sigma_{A=c}$ and $\sigma_{A=B}$, where $A, B \in U$ and $c \in \mathit{Dom}(A)$. These operators take as input relation I and return the following relations:

$$\sigma_{A=c}(I) = \{t \in I \mid t(A) = c\}, \qquad \sigma_{A=B}(I) = \{t \in I \mid t(A) = t(B)\}.$$

From now on, if $V \subseteq U$ and t is a U-tuple, then $t[V]$ denotes a V-tuple obtained by restricting t to V. In particular, for every attribute $A \in U$, $t[A]$ represents the value $t(A)$.

The general form of the projection operator is π_X, where $X \subseteq U$. On input I, this operator returns relation $\pi_X(I) = \{t[X] \mid t \in I\}$. Let $R'[U']$ be a relation schema and I' an instance of $R'[U']$. Assume that $V = U \cup U'$. Then, the join between I and I', denoted by $I \bowtie I'$, is defined as the following set of V-tuples:

$$\begin{aligned} I \bowtie I' \;=\; \{u \mid\ & u \text{ is a } V\text{-tuple and there exist } t \in I \text{ and} \\ & t' \in I' \text{ such that } u[U] = t \text{ and } u[U'] = t'\}. \end{aligned}$$

For example, by joining the relation shown in Figure 2.1 with

Room	*Location*
AN203	95 Queen's Park
GB258	35 St. George Street
LM161	80 St. George Street
GB248	35 St. George Street

we obtain the following relation:

Number	*Title*	*Section*	*Room*	*Location*
CSC 258	Computer Organization	2	GB258	35 St. George Street
CSC 258	Computer Organization	3	LM161	80 St. George Street
CSC 258	Computer Organization	3	GB248	35 St. George Street
CSC 434	Data Management Systems	1	GB248	35 St. George Street

Finally, union and difference operators are defined as the usual set theoretic operators. If J is an instance of $R[U]$, then $I \cup J = \{t \mid t \in I \text{ or } t \in J\}$ and $I - J = \{t \mid t \in I \text{ and } t \notin J\}$.

A relational algebra query, usually denoted by Q, is constructed by combining the relational algebra operators. For a database instance I, $Q(I)$ denotes the set of tuples obtained by executing query Q on I. For example, if I is the database instance shown in Figure 2.1, then $\pi_{Number,Title}(\sigma_{Room=\text{GB248}}(I)) = \{$(CSC 258, Computer Organization), (CSC 434, Data Management Systems)$\}$.

2.1.2 Data Dependencies in Relational Databases

In this section, we present some of the most popular data dependencies for relational databases: functional dependencies, key dependencies, multivalued dependencies, join dependencies and domain dependencies. These constraints play a central role in normalization theory for relational databases.

A common issue in every normalization algorithm (see Section 2.1.3 for a description of the most common normalization algorithms) is the use of dependency implication. Given a set of data dependencies $\Sigma \cup \{\sigma\}$, Σ implies σ, denoted by $\Sigma \models \sigma$, if for every database instance I that satisfy all the constraints in Σ, it is the case that I satisfies σ. The set of all dependencies implied by Σ is denoted by Σ^+. In this section, for each of the dependencies mentioned above we present algorithms for solving the implication problem.

$constraints := \Sigma$
$closure := X$
repeat until no further change:
    if $W \rightarrow Z \in constraints$ and $W \subseteq closure$ then
        $closure := closure \cup Z$
        $constraints := constraints - \{W \rightarrow Z\}$
return $closure$

Figure 2.2: An algorithm for computing the closure of a set of attributes X, given a set of functional dependencies Σ.

In this section, we also present an alternative approach to the implication problem. In this approach, inference rules are used to construct proofs that a dependency is implied. For a class of data dependencies $\mathcal{C}$, a set of inferences rules $\mathcal{I}$ is said to be complete if for every set of constraints $\Sigma \cup \{\sigma\}$ in $\mathcal{C}$, if $\Sigma \models \sigma$ then σ can be deduced from Σ by using the set of rules $\mathcal{I}$, denoted by $\Sigma \vdash_{\mathcal{I}} \sigma$. Furthermore, $\mathcal{I}$ is said to be sound if $\Sigma \vdash_{\mathcal{I}} \sigma$ implies that $\Sigma \models \sigma$. For each of the dependencies mentioned above, we show a finite, sound and complete set of inference rules, if such a set exists.

Functional and Key Dependencies

A *functional dependency* (FD) over a relation schema $R[U]$ is an expression of the form $X \rightarrow Y$, where $X, Y \subseteq U$. A relation I over $R[U]$ satisfies $X \rightarrow Y$, denoted by $I \models X \rightarrow Y$, if for every pair of tuples t_1, t_2 in I, $t_1[X] = t_2[X]$ implies $t_1[Y] = t_2[Y]$. Thus, $X \rightarrow Y$ says that if two tuples contain the same values on X, they must have the same values on Y. For example, the relation shown in Figure 2.1 satisfies the functional dependency *Number* $\rightarrow$ *Title*, since one title is associated to each course number. On the other hand, this relation does not satisfy the functional dependency *Number* $\rightarrow$ *Room*, because the first two tuples of this relation have the same value on the attribute *Number* and different values on the attribute *Room*.

A *key dependency* (KD) over $R[U]$ is a functional dependency of the form $X \rightarrow U$. If such a constraint exists, we say that X is a *superkey*. If there is no $Y \subsetneq X$ such that Y is a superkey, then X is a *key*. For instance, $\{$*Number*, *Section*, *Room*$\}$ is a key for the relation shown in Figure 2.1.

Let Σ be a set of functional dependencies over $R[U]$ and $X \subseteq U$. The closure of X, denoted by X^+, is defined to be the set of attributes $A \in U$ such that $\Sigma \models X \rightarrow A$. This set is used to determine whether a set of FDs implies a given FD; for every set of FDs $\Sigma \cup \{X \rightarrow Y\}$, $\Sigma \models X \rightarrow Y$ if and only if $Y \subseteq X^+$. The closure of a set of attributes X can be computed in quadratic time by using the algorithm shown in Figure 2.2. This algorithm incrementally computes the closure of X given a set of FDs Σ. In each of its iterations, at least one new attribute is added to *closure*, except for the last one. Thus, in the worst case the number of iterations is $|U|$. In each iteration, the algorithm needs to scan Σ and, therefore, in the worst case the algorithm runs in time $O(|U| \cdot \|\Sigma\|)$, where $\|\Sigma\|$ is the size of the representation[2] of Σ. Beeri and Bernstein [BB79, Ber79] proposed a linear time algorithm for computing the closure of a set of attributes. This algorithm is used to construct a linear time procedure for solving the implication problem for functional dependencies [BB79].

The implication problem for FDs is axiomatizable. The following is a sound and complete set of inference rules [Arm74]:

Reflexibility	:	If $Y \subseteq X$, then $X \rightarrow Y$.
Augmentation	:	If $X \rightarrow Y$, then $XZ \rightarrow YZ$,
Transitivity	:	If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

In these rules, we denote the union of two set of attributes X, Y by XY. Hereafter, we adopt this terminology.

Multivalued Dependencies

A *multivalued dependency* (MVD) over a schema $R[U]$ is an expression of the form $X \rightarrow\!\!\!\rightarrow Y$, where X and Y are subsets of U. A database instance I of $R[U]$ satisfies a multivalued dependency $X \rightarrow\!\!\!\rightarrow Y$, denoted by $I \models X \rightarrow\!\!\!\rightarrow Y$, if for every pair of tuples t_1, t_2 in $I(R)$ such that $t_1[X] = t_2[X]$, there exist a tuple t_3 in $I(R)$ such that $t_3[XY] = t_1[XY]$ and $t_3[XZ] = t_2[XZ]$, where $Z = U - XY$. Essentially, a database instance satisfies $X \rightarrow\!\!\!\rightarrow Y$ if for every value of X, the values in Y are independent of the values in Z, that is, a database instance I satisfies $X \rightarrow\!\!\!\rightarrow Y$ if and only if for every pair of X, Z-values x, z that appears in some tuple of I,

$$\{t[Y] \mid t \in I \text{ and } t[X] = x\} \quad = \quad \{t[Y] \mid t \in I,\ t[X] = x \text{ and } t[Z] = z\}.$$

[2] Observe that $\|\Sigma\|$ is $O(|U| \cdot |\Sigma|)$, where $|\Sigma|$ is the number of functional dependencies in Σ.

Theater	*Title*	*Snack*
Bloor Cinema	Bad Company	coffee
Bloor Cinema	Bad Company	popcorn
Bloor Cinema	Spider-Man	coffee
Bloor Cinema	Spider-Man	popcorn
Paramount	Bad Company	coke
Paramount	Bad Company	popcorn
Paramount	Insomnia	coke
Paramount	Insomnia	popcorn
Paramount	Spider-Man	coke
Paramount	Spider-Man	popcorn

Figure 2.3: Relation *Movie*.

For example, consider a relation schema $\mathit{Movie}(\mathit{Theater}, \mathit{Title}, \mathit{Snack})$ [AHV95]. A tuple (th, ti, sn) is in this relation if theater th is showing movie ti and offering snack sn. For a given theater, the information about titles and snacks is independent and, therefore, this schema must satisfy the MVD $\mathit{Theater} \twoheadrightarrow \mathit{Title}$. Figure 2.3 shows one instance of the relation *Movie* satisfying this multivalued dependency.

To solve the implication problem for multivalued dependencies, the concepts of "dependency set" and "dependency basis" were introduced by Beeri [Bee80]. Given a set of multivalued dependencies Σ over U and $X \subseteq U$, the *dependency set of* X, denoted by X^+, is defined as $\{Y \subseteq U \mid \Sigma \models X \twoheadrightarrow Y\}$. The dependency set is a generalization of the closure of a set attributes given a set of functional dependencies, defined in the previous section. Moreover, this collection is closed under union, intersection and difference. Thus, X^+ contains a unique sub-collection of nonempty, pairwise disjoint sets such that every element of X^+ is a union of some elements of this sub-collection. This set is called the *dependency basis of* X, denoted by $\mathit{dep}(X)$.

Given a set of multivalued dependencies $\Sigma \cup \{X \twoheadrightarrow Y\}$ over a set of attributes U, $\Sigma \models X \twoheadrightarrow Y$ if and only if $Y \in X^+$, that is, if and only if Y is the union of some elements of $\mathit{dep}(X)$. Thus, an algorithm for computing $\mathit{dep}(X)$ can be easily extended to solve the implication problem for multivalued dependencies. Such an algorithm is shown in Figure 2.4.

The algorithm shown in Figure 2.4 was proposed by Beeri [Bee80]. This algorithm incrementally stores in *basis* the dependency basis of X. Initially, this set contains the set of attributes that are trivially implied by X: A, for each $A \in X$, and $U - X$. Given $W \twoheadrightarrow Z \in \Sigma$, the inner loop construct a set of attributes W' such that $\Sigma \models X \twoheadrightarrow W'$ and $W \subseteq W'$. Hence, $\Sigma \models X \twoheadrightarrow W' - Z$ and, therefore, if $W' - Z$, denoted by Z' in the algorithm, is not empty and is not a union of some of the sets included in *basis*, then it is added to the dependency set. This is done in the last step of the algorithm. This algorithm runs in time $O(\|\Sigma\|^4)$ [Bee80].

The algorithm shown in Figure 2.4 constructs the dependency basis incrementally. In each step of the loop, it chooses one multivalued dependency and tries to refine *basis* by considering a subset of the right hand side of this dependency. This algorithm can be improved by considering a different refinement rule [Gal82]. Assume that $|\Sigma| = n$, the ith dependency in Σ is $W_i \twoheadrightarrow Z_i$ ($i \in [1, n]$) and Y is in *basis*, after executing k steps of the algorithm. If there is $i \in [1, n]$ such that $W_i \cap Y = \emptyset$, $Y \cap Z_i \neq \emptyset$ and $Y \cap (U - Z_i) \neq \emptyset$, then Y can be replaced by $Y_1 = Y \cap Z_i$ and $Y_2 = Y \cap (U - Z_i)$, since $\Sigma \models X \twoheadrightarrow Y \cap Z_i$ and $\Sigma \models X \twoheadrightarrow Y \cap (U - Z_i)$. The algorithm terminates when this refinement rule cannot be applied. Notice that if the multivalued dependency i is used to split a set Y into Y_1 and Y_2, then it can be used to split neither Y_1 nor Y_2, since $Y_1 \subseteq Z_i$ and $Y_2 \subseteq U - Z_i$. By using this idea, and a suitable data structure, an almost linear-time algorithm for computing the dependency basis was proposed by Galil [Gal82]. This algorithms runs in time $O(\min(|\Sigma|, \log |U|) \cdot \|\Sigma\|)$.

Usually, functional and multivalued dependencies have to be considered together in the normalization process. None of the algorithms presented so far can be directly used to solve the implication problem when these constraints are combined together. Fortunately, the algorithm shown above can be extended to solve this problem. Let Σ be a set of functional dependencies and multivalued dependencies. First, we show how to test whether a multivalued dependency is implied by Σ. For every $\varphi \in \Sigma$, define $M(\varphi)$ as follows. If φ is a functional dependency of the form $X \to Y$, then $M(\varphi)$ is a set of multivalued dependencies $\{X \twoheadrightarrow A \mid A \in Y\}$. If φ is a multivalued dependency, then $M(\varphi) = \{\varphi\}$. Observe that $\varphi \models M(\varphi)$. Furthermore, define $M(\Sigma)$ as the set of multivalued dependencies $\bigcup_{\varphi \in \Sigma} M(\varphi)$. It was shown by Beeri [Bee80] that Σ can be replaced by $M(\Sigma)$ in order to test whether a multivalued dependency σ is implied by Σ, that is, $\Sigma \models \sigma$ if and only if $M(\Sigma) \models \sigma$. Hence, it can be checked whether $\Sigma \models \sigma$ by computing $M(\Sigma)$ and then using the algorithm shown above. Second, we show how to test if a functional dependency is implied by Σ. Let $X \to A$ be a functional dependency

basis $:= \{\{A\} \mid A \in X\} \cup \{U - X\}$
change := true
while *change* do
  change := false
  for each $W \rightarrow\!\!\!\rightarrow Z \in \Sigma$ do
    $W' := \bigcup\{Y \mid Y \in \textit{basis} \text{ and } Y \cap W \neq \emptyset\}$
    $Z' := Z - W'$
    if $Z' \neq \emptyset$ and Z' is not the union of some element of *basis* then
      change := true
      basis := basis of the collection of boolean combinations of sets
        from $\textit{basis} \cup \{Z'\}$

Figure 2.4: Algorithm for computing $dep(X)$.

over U. Assume that $A \notin X$ and $dep(X)$ is the dependency basis of X with respect to $M(\Sigma)$. It was shown by Beeri [Bee80] that $\Sigma \models X \rightarrow A$ if and only if $\{A\} \in dep(X)$ and there is a nontrivial functional dependency in Σ with right hand side containing A. Hence, it can be checked whether $\Sigma \models X \rightarrow A$ by computing $dep(X)$ and checking whether $\{A\} \in dep(X)$ and there exists a functional dependency $W \rightarrow Z$ in Σ such that $A \in Z - W$.

Finally, we present axiomatizations for the implication problems for MVDs alone and MVDs combined with FDs. The following is a sound and complete set of inference rules for multivalued dependencies [BFH77, Men79]:

Complementation	:	If $X \rightarrow\!\!\!\rightarrow Y$, then $X \rightarrow\!\!\!\rightarrow (U - Y)$.
Reflexivity	:	If $Y \subseteq X$, then $X \rightarrow\!\!\!\rightarrow Y$.
Augmentation	:	If $X \rightarrow\!\!\!\rightarrow Y$, then $XZ \rightarrow\!\!\!\rightarrow YZ$.
Transitivity	:	If $X \rightarrow\!\!\!\rightarrow Y$ and $Y \rightarrow\!\!\!\rightarrow Z$, then $X \rightarrow\!\!\!\rightarrow (Z - Y)$.

Two rules have to be added to this set in order to have a sound and complete set of inference rules for functional and multivalued dependencies [BFH77]:

Conversion	:	If $X \rightarrow Y$, then $X \rightarrow\!\!\!\rightarrow Y$.
Interaction	:	If $X \rightarrow\!\!\!\rightarrow Y$ and $XY \rightarrow Z$, then $X \rightarrow (Z - Y)$.

Theater	*Title*	*Director*	*Snack*
Bloor Cinema	Bad Company	Joel Schumacher	coffee
Bloor Cinema	Bad Company	Joel Schumacher	popcorn
Bloor Cinema	Spider-Man	Sam Raimi	coffee
Bloor Cinema	Spider-Man	Sam Raimi	popcorn
Paramount	Bad Company	Joel Schumacher	coke
Paramount	Bad Company	Joel Schumacher	popcorn
Paramount	Insomnia	Christopher Nolan	coke
Paramount	Insomnia	Christopher Nolan	popcorn
Paramount	Spider-Man	Sam Raimi	coke
Paramount	Spider-Man	Sam Raimi	popcorn

Figure 2.5: Relation *MovieDirector*.

Join Dependencies

A join dependency (JD) over a relation schema $R[U]$ is an expression of the form $\bowtie[X_1, \ldots, X_n]$, where each X_i $(i \in [1, n])$ is a set of attributes and $X_1 \cup \cdots \cup X_n = U$. A database instance I of $R[U]$ satisfies $\bowtie[X_1, \ldots, X_n]$, denoted by $I \models \bowtie[X_1, \ldots, X_n]$, if $I = \pi_{X_1}(I) \bowtie \cdots \bowtie \pi_{X_n}(I)$ [3].

Multivalued dependencies are a special case of join dependencies consisting of two set of attributes; an MVD $X \rightarrow\!\!\!\rightarrow Y$ defined over a relation schema $R[U]$ is equivalent to $\bowtie[XY, X(U - XY)]$. In general, a join dependency can have an arbitrary arity. For example, consider the relation schema *MovieDirector*(*Theater*, *Title*, *Director*, *Snack*). A tuple (th, ti, di, sn) is in this relation if theater th is showing movie ti and offering snack sn, and the director of ti is di. Figure 2.5 shows one instance of the relation *MovieDirector*. This instance satisfies the join dependency $\bowtie[\{\mathit{Title}, \mathit{Director}\}, \{\mathit{Theater}, \mathit{Title}\}, \{\mathit{Theater}, \mathit{Snack}\}]$.

Usually, join dependencies, multivalued dependencies and functional dependencies are considered together in the normalization process. This gives rise to three different implication problems depending on whether the implicant is either an FD or an MVD or a JD. First, we show that if the implicant is an FD or an MVD then the implication problem can be solved in quadratic time. Second, we show that if the implicant is a

[3] We omit parenthesis in this expression since the join operator is associative.

JD then the implication problem is NP-hard. In this case, we also present a powerful tool for testing implication that in general requires exponential time and space. Finally, we present an efficient algorithm for testing implication for a natural subclass of join dependencies.

Let Σ be a set of functional dependencies, multivalued dependencies and join dependencies over a set of attributes U. For every $\varphi \in \Sigma$, defined $M(\varphi)$ as follows. If φ is either a functional dependency or a multivalued dependency, then $M(\varphi)$ is defined as in the previous section, that is, $M(X \rightarrow Y) = \{X \rightarrow\!\!\!\rightarrow A \mid A \in Y\}$ and $M(X \rightarrow\!\!\!\rightarrow Y) = \{X \rightarrow\!\!\!\rightarrow Y\}$. If φ is a join dependency of the form $\bowtie[X_1, \ldots, X_n]$, then $M(\varphi)$ is defined to be a set of multivalued dependencies [MSY81]:

$$\{\ T_I \cap T_J \rightarrow\!\!\!\rightarrow T_I \ \mid\ I, J \text{ is a partition of } \{1, \ldots n\},\ T_I = \bigcup_{i \in I} X_i \text{ and } T_J = \bigcup_{j \in J} X_j\ \}.$$

Observe that for each partition I, J of $\{1, \ldots, n\}$, $\varphi \models T_I \cap T_J \rightarrow\!\!\!\rightarrow T_I$. Define $M(\Sigma)$ as the set of multivalued dependencies $\bigcup_{\varphi \in \Sigma} M(\varphi)$. As it was seen in the previous section, if Σ contains only functional and multivalued dependencies, then $M(\Sigma)$ can be used to test in polynomial time whether Σ implies a functional dependency or a multivalued dependency. This result was extended by Maier et al. [MSY81] for the case of join dependencies. More precisely, it was proved by Maier et al. [MSY81] that for every multivalued dependency σ, $\Sigma \models \sigma$ if and only if $M(\Sigma) \models \sigma$, and for every nontrivial functional dependency σ of the form $X \rightarrow A$, $\Sigma \models \sigma$ if and only if $M(\Sigma) \models X \rightarrow\!\!\!\rightarrow A$ and there exists a nontrivial FD in Σ with right hand side containing A. Thus, as in the previous section, to test whether a functional dependency or a multivalued dependency with right hand side X is implied by Σ we only need to construct the dependency basis of X with respect to $M(\Sigma)$ and check some additional conditions. However, the dependency basis of a set of attributes X with respect to $M(\Sigma)$ cannot be directly computed by applying one of the algorithms shown in the previous section since the size of $M(\Sigma)$ is exponential in the size of Σ. An algorithm that constructs $dep(X)$ in time $O(|U| \cdot \|\Sigma\|)$, without materializing $M(\Sigma)$, was proposed by Maier et al. [MSY81]. A simplified version of this algorithm is presented next.

In Figure 2.4, we show an incremental algorithm for computing the dependency basis of a set of attributes X with respect to a set of multivalued dependencies Σ. In each step, this algorithm takes an approximation of the dependency basis (initially $\{A \mid A \in X\} \cup \{U - X\}$) and uses a multivalued dependency to refine it. The algorithm proposed by Maier et al. [MSY81] uses the same idea to calculate the dependency basis of X with respect to $M(\Sigma)$. In each step, this algorithm refines the current version of the dependency basis by using a multivalued dependency or a join dependency. If a multivalued dependency is chosen, then the refinement rule shown in Figure 2.4 is applied. Otherwise, a join dependency φ of the form $\bowtie[X_1, \ldots, X_n]$ is chosen and the following refinement rule is used. Let Y be a set of attributes in the last computed approximation of the dependency basis. Then, φ *refines* Y if there exists a partition I, J of $\{1, \ldots, n\}$ such that $Y \cap T_I \neq \emptyset$, $Y \cap T_J \neq \emptyset$ and $Y \cap T_I \cap T_J = \emptyset$. The algorithm verifies whether φ refines Y as follows. Define a graph $G(\varphi) = (Y, E)$, where $(A, B) \in E$ if there exists $i \in [1, n]$ such that $A, B \in X_i$. Then, φ refines Y if and only if G is disconnected [MSY81]. Furthermore, the connected components Y_1, …, Y_m of G form the refinement of Y. Thus, Y is replaced by Y_1, …, Y_m and the algorithm continues as shown in Figure 2.4.

Now, we turn our attention to the problem of verifying whether a join dependency σ is implied by a set Σ of FDs, MVDs and JDs. It was proved by Maier et al. [MSY81] and Fischer et al. [FT83] that this problem is NP-hard, even if either Σ contains one join dependency and no multivalued dependencies [MSY81] or Σ contains only multivalued dependencies [FT83]. To the best of our knowledge, the exact complexity of this problem remains open [FV86, AHV95]. Next, we present the best known algorithm for testing implication of join dependencies [MMS79].

The *chase* is a powerful tool for reasoning about dependencies. It was proposed by Aho et al. [ABU79] for testing implication of join dependencies by a set of functional dependencies, and it was extended by Maier et al. [MMS79] for reasoning about functional dependencies, multivalued dependencies and join dependencies. This tool requires exponential time and space for checking whether a JD is implied by a set of FDs, MVDs and JDs, although in general it is efficient enough to be used in practice. Furthermore, it is widely used in other problems like semantic query optimization [AHV95]. To present this tool we have to introduce some terminology.

A *tableau* is a set of rows with one column for each attribute in the universe U. The rows are composed of *distinguished* and *non-distinguished* variables. Each variable may appear in only one column and only one distinguished variable may appear in a column. For example, the following is a tableau for a universe with attributes A, B and C:

A	B	C
$\mathbf{x}$	$\mathbf{y}$	x_1
x_2	$\mathbf{y}$	$\mathbf{z}$
x_2	$\mathbf{y}$	x_3

In this case, $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$ are distinguished variables and x_1, x_2, x_3 are non-distinguished variables.

Assume that the non-distinguished variables in a tableau T are $x_1, \ldots, x_m$. The chase of T with respect to a set Σ of functional and join dependencies is based on the successive application of the following rules [MMS79]:

FD rule: Let σ be a functional dependency in Σ of the form $X \to A$, where A is a single attribute, and $u, v \in T$ be such that $u[X] = v[X]$ and $u[A] \neq v[A]$. The result of applying the FD σ to T is a new tableau T' defined as follows. If one of the variables $u[A], v[A]$ is distinguished, then all the occurrences of the other one are renamed to that variable. If both are non-distinguished, then all the occurrences of the variable with larger subscript are renamed to the variable with smaller subscript.

JD rule: Let σ be a join dependency of the form $\bowtie[X_1, \ldots, X_n]$ and u a tuple not in T. If there are $u_1, \ldots, u_n \in T$ such that $u_i[X_i] = u[X_i]$ for every $i \in [1, n]$, then the result of applying the JD σ over T is the new tableau $T' = T \cup \{u\}$.

A chasing sequence of T by Σ is a possibly infinite sequence of tableaux $T = T_0$, T_1, T_2, $\ldots$, such that for each $i \geq 0$, T_{i+1} is the result of applying some dependency in Σ to T_i. It was proved by Maier et al. [MMS79] that a given set of FD and JD rules can be applied to a tableau T only a finite number of times and, therefore, all these sequences are finite. Furthermore, it was shown in [MMS79] that if $T_0, \ldots, T_n$ and $T'_0, \ldots, T'_m$ are two terminal sequences generated from T (FD and JD rules can be applied neither to T_n nor to T'_m), then T_n and T'_m are equal. Thus, the chase of T by Σ, denoted by $Chase_\Sigma(T)$, is defined as the result of some terminal chasing sequence of T by Σ.

Every application of either the "FD rule" or the "JD rule" naturally defines a substitution of variables by variables (in the latter, this substitution is the identity). The substitution defined by the chase is obtained as the composition of the substitutions for each step of the chase. This substitution enables us to map each original variable (tuple) in T to a variable (tuple) in $Chase_\Sigma(T)$

Given a set of FDs and JDs $\Sigma \cup \{\sigma\}$, it was shown by Maier et al. [MMS79] that the chase can be used for checking whether $\Sigma \models \sigma$. The idea is to construct a tableau T_σ, compute $Chase_\Sigma(T_\sigma)$ and verify whether some condition is satisfied. If σ is a functional dependency of the form $X \to A$, then a tableau T_σ is constructed as follows. Tableau T_σ has two rows. The first row contains only distinguished variables and the second one contains distinguished variables in all the X-columns and non-distinguished variables in the remaining columns. It was proved in [MMS79] that $\Sigma \models \sigma$ if and only if $Chase_\Sigma(T_\sigma)$ has only distinguished variables in the A-column. For example, we can use the chase to check whether $\{\bowtie[AB, AC],\ AB \to C\} \models A \to C$, which corresponds to the interaction rule[4] defined for multivalued dependencies. In this case T_σ is equal to

A	B	C
$\mathbf{x}$	$\mathbf{y}$	$\mathbf{z}$
$\mathbf{x}$	x_1	x_2

A terminal sequence of tableaux is generated by using twice the JD $\bowtie[AB, AC]$ and once the FD $AB \to C$:

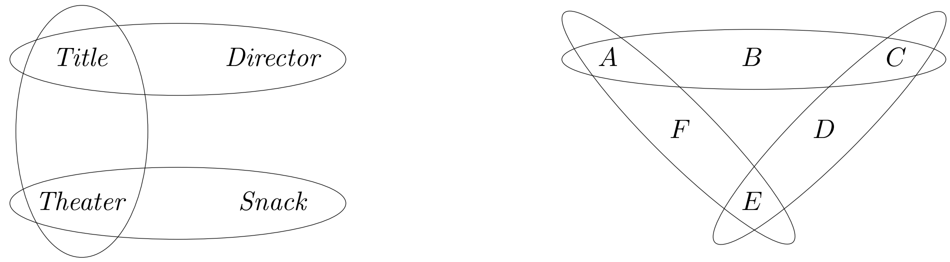
$$
\begin{array}{ccc} A & B & C \\ \hline \mathbf{x} & \mathbf{y} & \mathbf{z} \\ \mathbf{x} & x_1 & x_2 \\ & & \\ & & \end{array}
\xRightarrow{\bowtie[AB, AC]}
\begin{array}{ccc} A & B & C \\ \hline \mathbf{x} & \mathbf{y} & \mathbf{z} \\ \mathbf{x} & x_1 & x_2 \\ \mathbf{x} & \mathbf{y} & x_2 \\ & & \end{array}
\xRightarrow{\bowtie[AB, AC]}
\begin{array}{ccc} A & B & C \\ \hline \mathbf{x} & \mathbf{y} & \mathbf{z} \\ \mathbf{x} & x_1 & x_2 \\ \mathbf{x} & \mathbf{y} & x_2 \\ \mathbf{x} & x_1 & \mathbf{z} \end{array}
\xRightarrow{AB \to C}
\begin{array}{ccc} A & B & C \\ \hline \mathbf{x} & \mathbf{y} & \mathbf{z} \\ \mathbf{x} & x_1 & \mathbf{z} \\ & & \\ & & \end{array}
$$

The result of the chasing sequence is a tableau containing only distinguished variable $\mathbf{z}$ in the C-column. Therefore, $\{\bowtie[AB, AC],\ AB \to C\} \models A \to C$.

If σ is a join dependency of the form $\bowtie[X_1, \ldots, X_n]$, then a tableau T_σ is constructed as follows. Tableau T_σ has n rows. For every $i \in [1, n]$, the ith row contains distinguished variables in the X_i-columns and non-distinguished variables in the remaining columns. Furthermore, every non-distinguished variable in T_σ appears exactly once. It was shown by Maier et al. [MMS79] that $\Sigma \models \sigma$ if and only if $Chase_\Sigma(T_\sigma)$ has a row containing only distinguished variables.

The chase provides an exponential time algorithm for the implication problem for FDs, MVDs and JDs. It is desirable to find a subclass of join dependencies for which the implication problem is solvable in polynomial time. A natural subclass of join dependency satisfying this condition, and other desirable properties, was proposed by Fagin et al. [FMU82]. We briefly present this subclass next.

[4]Notice that $A \twoheadrightarrow B$ is represented by using join dependency $\bowtie[AB,\ AC]$.



(a) $\bowtie[\{\mathit{Title}, \mathit{Director}\}, \{\mathit{Theater}, \mathit{Title}\}, \{\mathit{Theater}, \mathit{Snack}\}]$ (b) $\bowtie[ABC, CDE, AEF]$

Figure 2.6: Hypergraphs of two join dependencies.

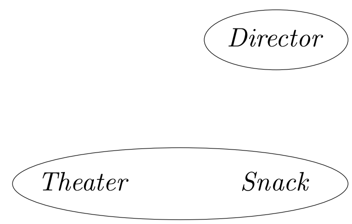
Every join dependency φ of the form $\bowtie[X_1, \ldots, X_n]$ induces a hypergraph $H(\varphi) = (U, E)$, where $U = \bigcup_{i=1}^{n} X_i$ and $E = \{X_i \mid i \in [1, n]\}$. For example, Figure 2.6 shows the hypergraphs induced from join dependencies $\bowtie[\{\mathit{Title}, \mathit{Director}\}, \{\mathit{Theater}, \mathit{Title}\}, \{\mathit{Theater}, \mathit{Snack}\}]$ and $\bowtie[ABC, CDE, AEF]$. It was shown by Fagin et al. [FMU82] that if the hypergraph of a JD φ satisfies some conditions, then φ can be replaced by a set of multivalued dependencies, and then the implication problem can be solved efficiently by using a polynomial time implication algorithm for multivalued dependencies. More precisely, it was proved by Fagin et al. [FMU82] that $H(\varphi)$ is acyclic if and only if φ is equivalent to a set if multivalued dependencies. It was also shown there that if φ is for the form $\bowtie[X_1, \ldots, X_n]$ and $H(\varphi)$ is acyclic, then φ is equivalent to the following set of multivalued dependencies:

$$\{X_i \cap X_j \twoheadrightarrow G \mid i, j \in [1, n] \text{ and } G \text{ is a connected component of } H(\varphi) \text{ with } X_i \cap X_j \text{ deleted}\}$$

This set contains $O(n^3)$ multivalued dependencies and it can be constructed in time $O(\|\varphi\|^3)$ by using a linear time algorithm for finding the connected components of a hypergraph. For example, join dependency $\bowtie[ABC, CDE, AEF]$ is cyclic and, therefore, not equivalent to any set of multivalued dependencies. On the other hand, join dependency $\bowtie[\{\mathit{Title}, \mathit{Director}\}, \{\mathit{Theater}, \mathit{Title}\}, \{\mathit{Theater}, \mathit{Snack}\}]$ is acyclic and equivalent to the following set of multivalued dependencies, obtained by using Fagin et al.'s algorithm [FMU82].

$$
\begin{array}{rclrcl}
\{\mathit{Title},\ \mathit{Director}\} & \twoheadrightarrow & \{\mathit{Theater},\ \mathit{Snack}\} & \mathit{Title} & \twoheadrightarrow & \mathit{Director} \\
\{\mathit{Theater},\ \mathit{Snack}\} & \twoheadrightarrow & \{\mathit{Title},\ \mathit{Director}\} & \mathit{Title} & \twoheadrightarrow & \{\mathit{Theater},\ \mathit{Snack}\} \\
\{\mathit{Title},\ \mathit{Theater}\} & \twoheadrightarrow & \mathit{Director} & \mathit{Theater} & \twoheadrightarrow & \{\mathit{Title},\ \mathit{Director}\} \\
\{\mathit{Title},\ \mathit{Theater}\} & \twoheadrightarrow & \mathit{Snack} & \mathit{Theater} & \twoheadrightarrow & \mathit{Snack} \\
\emptyset & \twoheadrightarrow & \{\mathit{Title},\ \mathit{Director},\ \mathit{Theater},\ \mathit{Snack}\} & & &
\end{array}
$$

For example, MVDs $\mathit{Title} \twoheadrightarrow \mathit{Director}$ and $\mathit{Title} \twoheadrightarrow \{\mathit{Theater}, \mathit{Snack}\}$ are obtained by removing from the hypergraph shown in Figure 2.6 (a) the set of attributes $\{\mathit{Title}\} = \{\mathit{Title}, \mathit{Director}\} \cap \{\mathit{Theater}, \mathit{Title}\}$:



and then computing the connected components of the generated hypergraph: $\{\mathit{Director}\}$ and $\{\mathit{Theater}, \mathit{Snack}\}$.

The set of multivalued dependencies shown above contains trivial and redundant dependencies. The Fagin et al. result was strengthened by Beeri et al. [BFMY83], who proved that acyclic join dependencies are equivalent to a linear-size set of multivalued dependencies. By using their technique, it is possible to show that $\bowtie[\{\mathit{Title}, \mathit{Director}\}, \{\mathit{Theater}, \mathit{Title}\}, \{\mathit{Theater}, \mathit{Snack}\}]$ is equivalent to $\{\mathit{Title} \twoheadrightarrow \mathit{Director}, \mathit{Theater} \twoheadrightarrow \mathit{Snack}\}$.

Finally, it is worth mentioning that Petrov [Pet89] proved that there is no a finite, sound and complete system of inference rules for join dependencies.

Inclusion and Foreign Key Dependencies

An *inclusion dependency* (ID) over a database schema $S = \{R_1[U_1], \ldots, R_n[U_n]\}$ is an expression of the form:

$$R_k[A_1, \ldots, A_m] \ \subseteq\ R_l[B_1, \ldots, B_m],$$

where $k, l \in [1, n]$, $\{A_1, \ldots, A_m\} \subseteq U_k$ and $\{B_1, \ldots, B_m\} \subseteq U_l$. A relation I over S satisfies this constraint if for every tuple t in $I(R_k)$, there exists a tuple t' in $I(R_l)$ such that $t[A_i] = t'[B_i]$ for every $i \in [1, m]$. Thus, $R_k[A_1, \ldots, A_m] \subseteq R_l[B_1, \ldots, B_m]$ says that $\pi_{A_1,\ldots,A_m}(I(R_k))$ is contained in $\pi_{B_1,\ldots,B_m}(I(R_l))$. For example, Figure 2.7 shows an

Employee_Number	*Name*	*Manager_Number*
99900	John Smith	99000
99901	Peter Levene	99000
99910	John Fox	99901
99920	Michael Myers	99901
99930	Steven Lockwood	99901

Figure 2.7: A database instance storing information about employees and their managers.

instance I of relation schema $R[Employee\_Number, Name, Manager\_Number]$ for storing information about employees and their managers. This instance satisfies the inclusion dependency $R[Manager\_Number] \subseteq R[Employee\_Number]$ saying that every manager is also an employee. We note that the CEO of the company is John Smith since he is his own manager.

A *foreign key dependency* (FKD) over a database schema $S = \{R_1[U_1], \ldots, R_n[U_n]\}$ is an expression of the form:

$$R_k[A_1, \ldots, A_m] \quad \subseteq_{FK} \quad R_l[B_1, \ldots, B_m],$$

where $k, l \in [1, n]$, $\{A_1, \ldots, A_m\} \subseteq U_k$ and $\{B_1, \ldots, B_m\} \subseteq U_l$. A relation I over S satisfies this constraint if I satisfies inclusion dependency $R_k[A_1, \ldots, A_m] \subseteq R_l[B_1, \ldots, B_m]$ and $B_1, \ldots, B_m$ is a superkey for $I(R_l)$, that is, $I(R_l) \models \{B_1, \ldots, B_m\} \rightarrow U_l$. Thus, the foreign key dependency shown above is equivalent to an inclusion dependency together with a key dependency. We note that the instance I shown in Figure 2.7 satisfies the foreign key dependency $R[Manager\_Number] \subseteq_{FK} R[Employee\_Number]$ since I satisfies inclusion dependency $R[Manager\_Number] \subseteq R[Employee\_Number]$ and *Employee_Number* is a superkey for this instance.

Casanova et al. [CFP84] showed that the implication problem for inclusion dependencies is PSPACE-complete. In this paper, the authors also showed that the following is a sound and complete set of inference rules for the implication problem:

Reflexibility	:	$R[X] \subseteq R[X]$,
Projection and permutation	:	If $R[A_1, \ldots, A_m] \subseteq S[B_1, \ldots, B_m]$, then $R[A_{i_1}, \ldots, A_{i_k}] \subseteq S[B_{i_1}, \ldots, B_{i_k}]$ for each sequence $i_1, \ldots, i_k$ of distinct integers from $\{1, \ldots, m\}$.
Transitivity	:	If $R[X] \subseteq S[Y]$ and $S[Y] \subseteq T[Z]$, then $R[X] \subseteq T[Z]$.

Mitchell and Chandra et al. [Mit83, CV85] independently showed that the implication problem for inclusion and functional dependencies taken together is undecidable. Casanova et al. [CFP84] also proved that there is no a finite, sound and complete system of inference rules for inclusion and functional dependencies taken together.

In light of these negative results, Cosmadakis et al. [CKV90] investigated the complexity of the implication problem for some restricted classes of inclusion dependencies. More specifically, they defined unary inclusion dependencies as IDs of the form $R[A] \subseteq S[B]$, that is, inclusion dependencies mentioning only one attribute in each side, and then they proved that the implication problem for functional dependencies and unary inclusion dependencies taken together is decidable in cubic time.

An interesting variation of the implication problem is obtained by considering only key and foreign key dependencies. This variation has been used recently to prove the undecidability of some problems involving XML data dependencies (see Chapter 5). To the best of our knowledge, the undecidability of the implication problem for key and foreign key dependencies was shown to be undecidable only recently by Fan and Siméon [FS00]. Fan and Libkin [FL01] also considered this problem and showed an stronger result, namely that the implication problem for key dependencies by key and foreign key dependencies is undecidable.

Domain Dependencies

So far, we have assumed that the domain of an attribute is infinite. However, it is easy to find examples where an attribute can take a finite set of values. For instance, the value of an attribute *Gender* can be either *male* or *female*.

Database management systems allow a user, by means of the SQL statement `CREATE TABLE`, to specify attributes with a finite domain. On the theory side, it is possible to specify finite domains, and in general restrictions on the domains, by using domain dependencies [Fag81]. A domain dependency over a database schema S is a constraint of the form $IN(A, D)$, where A is an attribute in S and $D \subseteq Dom(A)$. A database instance

I of S satisfies $IN(A, D)$, denoted by $I \models IN(A, D)$, if every value in an A-column is in D. For example, the finite domain shown in the previous paragraph can be specified by using the domain dependency $IN(Gender, \{male, female\})$.

All the implication algorithms presented in the previous sections become incomplete if domain dependencies are added. For instance, in a relation schema $R(A, B)$, from the domain dependency $IN(B, \{1\})$ is possible to infer that $A \rightarrow B$. The FD implication algorithm presented in this section is not able to deduce this functional dependency.

2.1.3 Relational Databases Design

Codd [Cod72] showed that a database containing functional dependencies may exhibit some anomalies when the information is updated. For example, consider the university database schema *Course*(*Number*, *Title*, *Section*, *Room*) presented in Section 2.1.1. The specification of this database includes functional dependency *Number* $\rightarrow$ *Title* since each course has only one title. Figure 2.8 shows one instance of this database. This instance is prone to three different types of anomalies. First, if the name of the course with number CSC 258 is changed to Computer Organization I, then four distinct cells need to be updated. If any of them is not updated, then the information in the database becomes inconsistent. This anomaly was called an *update anomaly* by Codd [Cod72] and it arises because the instance is storing redundant information. Second, if the information is updated because a new semester is starting, and the course with number CSC 434 is not given in that semester, then the last tuple of the instance is deleted and no information about CSC 434 appears in the updated instance. But this has the additional effect of deleting the title of the course, which will be the same the next time that CSC 434 is offered. This anomaly was called a *deletion anomaly* by Codd [Cod72] and it arises because the relation is storing information that is not directly related: The sections of a course vary from one term to another while its title is likely not to be changed from one semester to the next one. This can also lead to *insertion anomalies* [Cod72]; if a new course (CSC 336, Numerical Methods) is created, then it cannot be added to the database until at least one section and one room is assigned to the course.

To avoid updates anomalies, Codd introduced two normal forms [Cod72]. Each of these forms specifies some syntactic properties that the set of functional dependencies in a database must satisfy. For example, the database shown in Figure 2.8 is prone to update anomalies since the attribute *Title* partially depends on the key of the relation

Number	*Title*	*Section*	*Room*
CSC 258	Computer Organization	1	LP266
CSC 258	Computer Organization	2	GB258
CSC 258	Computer Organization	3	LM161
CSC 258	Computer Organization	3	GB248
CSC 434	Data Management Systems	1	GB248

Figure 2.8: A database prone to update anomalies.

{*Number*, *Section*, *Room*}. Moreover, Codd [Cod72] informally showed how to transform a database to generate a schema satisfying these normal forms. For instance, the database schema shown in the previous paragraph should be split into two relation schemas, namely *CourseName*(*Number*, *Title*) and *Course*(*Number*, *Section*, *Room*), to avoid the anomalies presented above.

In this section, we present the most popular normal forms: 3NF, BCNF, 4NF, PJ/NF, 5NFR and DK/NF. These normal forms were introduced to deal with functional dependencies (3NF, BCNF), multivalued dependencies (4NF), join dependencies (PJ/NF, 5NFR) and data dependencies in general (DK/NF). For each of these normal forms, we present algorithms for testing whether a given database schema satisfies a normal form and for transforming a database schema into a new one conforming to a normal form. The latter algorithms have been called *normalization algorithms* in the literature, and they involve transformation of schemas. Two basic properties have been used to test their correctness: information losslessness and dependency preservation. These properties are presented in detail in the first subsection of this section.

Schema Transformation

A normalization algorithm takes as input a relation schema and generate a database schema in some particular normal form. It is desirable that these two are as similar as possible, that is, they should contain the same data and the same semantic information. These properties have been called *information losslessness* and *dependency preservation* in the literature, respectively. We introduce them next.

Let S_1, S_2 be two database schemas. Intuitively, two instances I_1 of S_1 and I_2 of S_2 contain the same information if it is possible to retrieve the same information from them, that is, for every query Q_1 over I_1 there exists a query Q_2 over I_2 such that

$Q_1(I_1) = Q_2(I_2)$, and vice versa. To formalize this notion one needs to choose a query language. If this query language is relational algebra, then this notion is captured by the notion of calculously dominance introduced by Hull [Hul86]. Schema S_2 *dominates* S_1 *calculously* if there exist relational algebra expressions Q over S_1 and Q' over S_2 satisfying the following property: For every instance I of S_1, there exists an instance I' of S_2 such that $Q(I) = I'$ and $Q'(I') = I$. Thus, every query Q_1 over I can be transformed into an equivalent query $Q_2 = Q_1 \circ Q'$ over I', since $Q_2(I') = Q_1(Q'(I')) = Q_1(I)$, and, analogously, every query Q_2 over I' can be transformed into an equivalent query $Q_1 = Q_2 \circ Q$ over I, since $Q_1(I) = Q_2(Q(I)) = Q_2(I')$.

Normalization algorithms try to achieve the goal of information losslessness; if any of them transforms a database schema S into a database schema S', then S' should dominate S calculously. All the normalization algorithms presented in this section use only the projection operator to transform a schema[5] and, thus, calculously dominance is defined in terms of this operator and its inverse, the join operator. More precisely, the normalization algorithms presented in this section take as input a relation schema $S = (R[U], \Sigma)$ and use the projection operator to transform it into a database schema $S' = \{(R_i[U_i], \Sigma_i) \mid i \in [1, n]\}$ in some normal form. Then, S' is a *lossless decomposition* of S if for every instance I of S there is an instance I' of S' such that [ABU79]:

1. For every $i \in [1, n]$, $I'(R_i) = \pi_{U_i}(I)$.

2. $I = I'(R_1) \bowtie I'(R_2) \bowtie \cdots \bowtie I'(R_n)$.

That is, every instance I of S can be transformed into an instance I' of S' by using the projection operator, and I can be reconstructed from I' by using the join operator. It is straightforward to prove that S' is a lossless decomposition of S if and only if $\Sigma \models \bowtie[U_1, \ldots, U_n]$.

Let S, S' be as above. We define here the concept of dependency preservation for functional dependencies (for multivalued dependencies, this concept is introduced later). It is straightforward to prove that if $X, Y \subseteq V \subseteq U$ and I is an instance of S, then $I \models X \to Y$ if and only if $\pi_V(I) \models X \to Y$. Hence, $\bigcup_{i=1}^{n} \Sigma_i$ can be considered as a set of

[5]These algorithms have been called vertical decomposition algorithms in the literature [PBGG89], as opposed to horizontal decomposition algorithms. In the horizontal decomposition approach, the database schema contains functional dependencies and *afunctional dependencies* [PBGG89], and it is decomposed by considering some goals. The decomposition is achieved by using a relation algebra expression containing projection, selection and join operators among others. For a survey on horizontal decomposition see [PBGG89].

constraints over S. We use this property in the definition of dependency preservation; we say that S' is a *dependency preserving decomposition* of S if and only if $(\bigcup_{i=1}^{n} \Sigma_i)^+ = \Sigma^+$, that is, $\bigcup_{i=1}^{n} \Sigma_i$ and Σ are equivalent as sets of FDs over S.

Third Normal Form (3NF)

In order to avoid update anomalies in database schemas containing functional dependencies, Codd [Cod72] introduced two normal forms: second normal form (2NF) and third normal form (3NF). In this section, we only consider 3NF since every schema that is in 3NF is also in 2NF.

Let $R[U]$ be a relation schema and Σ a set of functional dependencies over $R[U]$. We say that an attribute A is a *prime* attribute if A is an element of some key of $R[U]$, and we say that $(R[U], \Sigma)$ is in 3NF if for every nontrivial functional dependency $X \to A \in \Sigma^+$, X is a superkey or A is a prime attribute[6]. Furthermore, we say that a database schema S is in 3NF if every relation schema in S is in 3NF. For example, relation schema $(\mathit{Course}(\mathit{Number}, \mathit{Title}, \mathit{Section}, \mathit{Room}), \{\mathit{Number} \to \mathit{Title}\})$ is not in 3NF since *Number* is not a superkey and *Title* is not a prime attribute. On the other hand, database schema $\{(\mathit{CourseName}(\mathit{Number}, \mathit{Title}), \{\mathit{Number} \to \mathit{Title}\}), (\mathit{Course}(\mathit{Number}, \mathit{Section}, \mathit{Room}), \emptyset)\}$ is in 3NF since *Number* is a superkey in relation *CourseName*.

For every normal form two problems have to be addressed: How to decide whether a schema is in that normal form, and how to transform a schema into an equivalent one in that normal form. In the rest of this section, we address these problems for the case of 3NF.

Unfortunately, it is expensive to check whether a schema is in 3NF. It was shown by Jou and Fischer that this problem is NP-complete [JF82]. Interestingly, in real life examples, it is usually not that expensive to check this condition. We present here an algorithm introduced by Mannila and Räihä [MR89] for testing if an attribute is prime, and we combine it with a Lemma of Jou and Fischer [JF82] to obtain a procedure for testing whether a relation schema is in 3NF. This algorithm works in polynomial time in the number of *maximal sets* not determining an attribute. Mannila and Räihä [MR89] showed some theoretical and practical evidence that this quantity is small in practice and exponential only for some "pathological" schemas, so that this algorithm can be used in

[6] This definition was proposed by Zaniolo [Zan82] and is equivalent to the original definition given by Codd [Cod72].

real life databases.

First, we present Mannila and Räihä's algorithm for testing primality. Let $R[U]$ be a relation schema and Σ a set of FDs over U. For every $A \in U$, define $max(A)$ as follows [MR89].

$$max(A) = \{Y \subseteq U \mid Y \text{ is a maximal set with respect to set inclusion such that } Y \rightarrow A \notin \Sigma^+\}.$$

Furthermore, define $max(U)$ as $\bigcup_{A \in U} max(A)$. It can be verified whether $X \in max(A)$ in time $O(|U| \cdot \|\Sigma\|)$ by using the following condition: $X \in max(A)$ if and only if $X \rightarrow A \notin \Sigma^+$ and $XB \rightarrow A \in \Sigma^+$, for every $B \in U - XA$.

It was proved by Mannila and Räihä [MR89] that an attribute A is prime if and only if there exists $X \in max(A)$ such that XA is a superkey. Thus, it can be verified whether A is a prime attribute in time $O(|max(A)| \cdot \|\Sigma\|)$, if the set $max(A)$ is given. Furthermore, this algorithm can be used to verify whether a relation schema is in 3NF. Let $R[U]$ be as above. An attribute $A \in U$ is *abnormal* if there exists $X \subseteq U$ such that $X \rightarrow A$ is a nontrivial functional dependency in Σ^+ and X is not a superkey [JF82]. It was proved by Jou and Fischer [JF82] that an attribute A is abnormal if and only if there exists $X \rightarrow Y \in \Sigma$ such that $A \in Y - X$ and X is not a superkey, and, therefore, the set of abnormal attributes of U can be computed in time $O(\|\Sigma\|^2)$ by using a linear time algorithm for computing the closure of a set of attributes (see Section 2.1.2). Moreover, relation schema $R[U]$ is in 3NF if and only if every abnormal attribute in U is prime, and, therefore, if $max(A)$ is given for every $A \in U$, then it can be tested in time $O(\|\Sigma\|^2 + \|\Sigma\| \cdot \sum_{A \in U} |max(A)|)$ whether $R[U]$ is in 3NF. Thus, if $max(A)$ has been precomputed, for every $A \in U$, then it can be checked in quadratic time whether $(R[U], \Sigma)$ is in 3NF. An interesting corollary of this is that given a relation schema $R[U]$ and a set Σ of unary functional dependencies over $R[U]$ (FDs of the form $A \rightarrow B$, where A, B are attributes), it can be tested in polynomial time whether $(R[U], \Sigma)$ is in 3NF since in this case $|max(A)| = 1$, for every $A \in U$.

Now, we turn our attention to the problem of decomposing a relation schema into a new schema in 3NF. Fortunately, for every relation schema S there is a database schema S' such that S' is in 3NF and S' is a lossless and dependency preserving decomposition of S. Furthermore, schema S' can be generated efficiently by using the synthesis approach introduced by Bernstein et al. [Ber76, BDB79]. To present this approach, we need to introduce some terminology.

set $S' := \emptyset$
find a minimal cover Γ of Σ
find a LHS-partition $\Gamma_1, \ldots, \Gamma_n$ of Γ
$S' := \{(R_i[U_i], \Gamma_i) \mid U_i$ is the set of all attributes appearing in $\Gamma_i\}$
if there is $(R_i[U_i], \Gamma_i)$ such that U_i is a superkey
  then output S'
  else
    determine a key X of U
    output $S' \cup \{(R_{n+1}[X], \emptyset)\}$

Figure 2.9: An algorithm for synthesizing 3NF schemas.

Given a set of functional dependencies Σ, a *minimal cover* of Σ is a set functional dependencies Γ such that: (1) $\Sigma^+ = \Gamma^+$; (2) no proper subset of Γ is equivalent to Σ; and (3) for each $X \to Y \in \Gamma$, there is no $Z \subsetneq X$ such that $Z \to Y \in \Gamma^+$. A partition $\Sigma_1, \ldots, \Sigma_n$ of Σ is a *LHS-partition* of Σ if all functional dependencies in Σ_i ($i \in [1, n]$) have the same left hand side, and no two sets Σ_i, Σ_j ($i \neq j$) have the same left hand side.

An algorithm for producing dependency preserving 3NF decompositions was introduced by Bernstein [Ber76], and it was extended by Biskup et al. [BDB79] to generate lossless and dependency preserving decompositions. Figure 2.9 shows this algorithm. The input of this procedure is a relation schema $(R[U], \Sigma)$ and it requires quadratic time to output a database schema S' in 3NF, since a minimal cover of a set of functional dependencies can be found in quadratic time [BB79].

If we apply the synthesis algorithm shown in Figure 2.9 to $(\mathit{Course}(\mathit{Number}, \mathit{Title}, \mathit{Section}, \mathit{Room}), \{\mathit{Number} \to \mathit{Title}\})$, we obtain the desired database schema $\{(\mathit{CourseName}(\mathit{Number}, \mathit{Title}), \{\mathit{Number} \to \mathit{Title}\}), (\mathit{Course}(\mathit{Number}, \mathit{Section}, \mathit{Room}), \emptyset)\}$. Observe that if we eliminate the last if-statement from this algorithm, then we obtain only the first relation schema $(\mathit{CourseName}(\mathit{Number}, \mathit{Title}), \{\mathit{Number} \to \mathit{Title}\})$, which is a dependency preserving decomposition of the original schema, but it is not a lossless decomposition. This last if-statement was included by Biskup et al. [BDB79] to ensure information losslessness.

Boyce-Codd Normal Forms (BCNF)

In general, a schema in 3NF is considered to be well designed. However, in some cases a 3NF relation can be prone to update anomalies. For instance, consider a relation $Code(Address, City, PostalCode)$ containing FDs $\{Address, City\} \to PostalCode$ and $PostalCode \to City$. Figure 2.10 shows an instance of this schema. Observe that $Address \to PostalCode$ is not a valid dependency in this schema since the same address can be associated with different postal codes in different cities, and $PostalCode \to Address$ is not a valid dependency because many addresses can share the same postal code. This schema is prone to update anomalies, although it is in 3NF since $\{Address, City\}$ is a key and $City$ is a prime attribute. For example, if Ottawa was incorrectly associated to K1S 5B6 and it has to be changed to London, then two cells have to be updated.

Address	*City*	*PostalCode*
10 King's College Road	Toronto	M5S 3G4
10 King's College Road	Ottawa	K1S 5B6
32 King's College Road	Ottawa	K1S 5B6

Figure 2.10: A 3NF relation prone to an update anomaly.

To avoid the kind of update anomalies shown in Figure 2.10, a more restrictive normal form, which eliminates the distinction between prime and non-prime attributes, was introduced by Codd[7] [Cod74]. Let $R[U]$ be a relation schema and Σ be a set of functional dependencies over $R[U]$. Then, $(R[U], \Sigma)$ is in Boyce Codd Normal Form (BCNF) [Cod74] if for every nontrivial functional dependency $X \to A \in \Sigma^+$, X is a superkey. Furthermore, a database schema S is in BCNF if every relation schema in S is in BCNF. For instance, the relation schema shown in the previous paragraph is not in BCNF since $PostalCode \to City$ is a nontrivial functional dependency in this schema and $PostalCode$ is not a superkey.

As opposed to the case of 3NF, it can be tested efficiently whether a relation schema is in BCNF. A relation schema $(R[U], \Sigma)$ is in BCNF if and only if for every nontrivial functional dependency $X \to Y \in \Sigma$, $X \to U \in \Sigma^+$. Thus, it is possible to check in quadratic time whether $(R[U], \Sigma)$ is in BCNF by using the linear time algorithm for functional dependency implication developed by Beeri and Bernstein [BB79, Ber79].

[7] Codd pointed out in [Cod74] that this normal form was developed by Raymond F. Boyce and himself.

set $S' := \{(R[U], \Sigma)\}$

repeat until S' is in BCNF

  choose a relation schema $(R'[U'], \Sigma') \in S'$ that is not in BCNF

  choose nonempty disjoint set of attributes X, Y, Z such that

    $XYZ = U'$, $\Sigma' \models X \to Y$ and $\Sigma' \not\models X \to A$, for every $A \in Z$

  replace $(R'[U'], \Sigma')$ by $(R_1[XY], \pi_{XY}(\Sigma'))$ and $(R_2[XZ], \pi_{XZ}(\Sigma'))$,

  where R_1 and R_2 are fresh relation names.

Figure 2.11: An algorithm for generating BCNF schemas [AHV95].

On the other hand, given a relation schema S, it is not always possible to find a database schema S' such that S' is in BCNF and S' is a lossless and dependency preserving decomposition of S. For instance, by constructing all possible lossless decompositions of the relation schema $(\mathit{Code}(\mathit{Address}, \mathit{City}, \mathit{PostalCode}),\ \{\mathit{PostalCode} \to \mathit{City}, \{\mathit{Address}, \mathit{City}\} \to \mathit{PostalCode}\})$, it is possible to prove that this schema does not admit a dependency preserving decomposition in BCNF. In general, for every relation schema $(R[U], \Sigma)$ there exists a database schema S' such that S' is in BCNF and S' is a lossless decomposition of S. This decomposition can be constructed by using the algorithm shown in Figure 2.11. In this algorithm, $\pi_X(\Sigma)$ represents the projection of a set of functional dependencies Σ over a set of attributes X, that is, $\{Y \to Z \mid \Sigma \models Y \to Z \text{ and } YZ \subseteq X\}$.

We note that in the worst case the previous algorithm runs in exponential time and space, since $|\pi_X(\Sigma)|$ can be exponential in the size of Σ. A possible solution to this problem is to replace $\pi_X(\Sigma)$ by an equivalent set of functional dependencies of polynomial size. Unfortunately, there are cases where such a set does not exists. Furthermore, it was proved by Beeri and Bernstein [BB79] that given a relation schema $(R[U], \Sigma)$ and $V \subseteq U$, the problem of verifying whether $(R[V], \pi_V(\Sigma))$ is in BCNF is coNP-complete. Thus, the algorithm shown in Figure 2.11 cannot run in polynomial time, even if $\pi_X(\Sigma)$ is not materialized, unless $\mathrm{P} = \mathrm{NP}$.

In general, unless $\mathrm{P} = \mathrm{NP}$, there is no an efficient algorithm for constructing a lossless and dependency preserving BCNF decomposition of a relation schema, if such a decomposition exists, since the problem of verifying whether a relation schema admits a lossless and dependency preserving decomposition into BCNF is coNP-hard [BB79]. Interestingly, a polynomial time algorithm for finding some BCNF decomposition was developed by Tsou and Fischer [TF82]. Let S be a relation schema $(R[U], \Sigma)$. Tsou and Fischer showed that if any of the following conditions holds, then $(R[U], \Sigma)$ is in BCNF:

(a) $|U| \leq 2$.

(b) $|U| > 2$ and for any pair of distinct attributes $A, B \in U$, $\Sigma \not\models U - AB \rightarrow A$.

Thus, if $|U| > 2$ and $(R[U], \Sigma)$ does not satisfy condition (b), then Tsou and Fischer's algorithm extracts from U a set of attributes U_1 such that $(R_1[U_1], \pi_{U_1}(\Sigma))$ satisfies this condition. This is achieved by using the following algorithm:

set $U_1 := U$
for each attribute $A \in U_1$ do
  for each attribute $B \in U_1 - A$ do
    if $\Sigma \models U_1 - AB \rightarrow A$ then $U_1 := U_1 - B$ and $C := A$
$U_2 := U - \{C\}$

The same algorithm is applied to the remaining set of attributes U_2, until $|U_2| \leq 2$ or $(R_2[U_2], \pi_{U_2}(\Sigma))$ satisfies condition (b). For instance, the schema that we have been using in this section does not satisfies condition (b) since $\{\mathit{Address}, \mathit{City}, \mathit{PostalCode}\} - \{\mathit{City}, \mathit{Address}\} \rightarrow \mathit{City}$. Hence, $\{\mathit{Address}$, City, $\mathit{PostalCode}\}$ is split into two set of attributes $\{\mathit{PostalCode}$, $\mathit{City}\}$ and $\{\mathit{Address}$, $\mathit{PostalCode}\}$, each of them satisfying condition (a). We note that this algorithm runs in polynomial time, since it does not need to compute $\pi_{U_2}(\Sigma)$ in order to check whether U_2 satisfies either (a) or (b). Indeed, it was shown by Tsou and Fischer that it requires time $O(|U|^4 \cdot \|\Sigma\|)$ to compute a BCNF decomposition.

Fourth Normal Form (4NF)

A database schema containing multivalued dependencies can be also prone to update anomalies, as Fagin pointed out in [Fag77]. For instance, consider again the relation schema *Movie*(*Theater*, *Title*, *Snack*), introduced in Section 2.1.2, containing multivalued dependency $\mathit{Theater} \twoheadrightarrow \mathit{Title}$. Figure 2.12 (a) shows one instance of this schema. If we want to insert tuple (Bloor Cinema, Bad Company, coke) into this relation, then we also have to insert tuple (Bloor Cinema, Spider-Man, coke) into it, since the updated relation has to satisfy MVD $\mathit{Theater} \twoheadrightarrow \mathit{Title}$. We note that a database containing only functional dependencies is not prone to this type of insertion anomalies; we cannot

Theater	*Title*	*Snack*
Bloor Cinema	Bad Company	coffee
Bloor Cinema	Bad Company	popcorn
Bloor Cinema	Spider-Man	coffee
Bloor Cinema	Spider-Man	popcorn

(a) Original relation.

Theater	*Title*	*Snack*
Bloor Cinema	Bad Company	coffee
Bloor Cinema	Bad Company	popcorn
Bloor Cinema	Bad Company	coke
Bloor Cinema	Spider-Man	coffee
Bloor Cinema	Spider-Man	popcorn
Bloor Cinema	Spider-Man	coke

(b) Updated relation.

Figure 2.12: A new type of insertion anomaly.

solve an insertion anomaly in a database containing functional dependencies by inserting additional tuples.

To avoid the type of anomalies shown in Figure 2.12, Fagin introduced a normal form for functional and multivalued dependencies. Let $R[U]$ be a relation schema and Σ a set of FDs and MVDs. Then $(R[U], \Sigma)$ is in fourth normal form (4NF) if for every nontrivial multivalued dependency $X \twoheadrightarrow Y$ implied by Σ, X is a superkey. Moreover, a database schema S is in 4NF if every relation schema in S is in 4NF. For example, the relation schema shown in the previous paragraph is not in 4NF since *Theater* is not a superkey. Observe that if Σ contains only functional dependencies and $(R[U], \Sigma)$ is in 4NF, then for every nontrivial FD $X \rightarrow A$ implied by Σ, $X \twoheadrightarrow A$ is a nontrivial MVD and, therefore, X is a superkey. Thus, $(R[U], \Sigma)$ is in BCNF.

Analogously to the case of functional dependencies, it is possible to prove that a relation schema $(R[U], \Sigma)$ is in 4NF if and only if for every nontrivial FD $X \rightarrow Y \in \Sigma$, $\Sigma \models X \rightarrow U$, and for every nontrivial MVD $X \twoheadrightarrow Y \in \Sigma$, $\Sigma \models X \rightarrow U$. Thus, by using Galil's algorithm [Gal82] for testing implication of multivalued dependencies, it can be verified in almost quadratic-time $O(n^2 \cdot \log n)$ whether a relation schema is in 4NF.

Before turning our attention to the problem of 4NF decomposition, we need to introduce the concept of dependency preservation for multivalued dependencies. Consider again the relation schema $(\mathit{Movie}(\mathit{Theater}, \mathit{Title}, \mathit{Snack}), \{\mathit{Theater} \twoheadrightarrow \mathit{Title}\})$. A natural 4NF decomposition of this schema is

$$\{(\mathit{Movie}_1(\mathit{Theater}, \mathit{Title}), \{\mathit{Theater} \twoheadrightarrow \mathit{Title}\}),$$
$$(\mathit{Movie}_2(\mathit{Theater}, \mathit{Snack}), \{\mathit{Theater} \twoheadrightarrow \mathit{Snack}\})\}. \quad (2.1)$$

Observe that *Theater* $\rightarrow\!\!\rightarrow$ *Title* and *Theater* $\rightarrow\!\!\rightarrow$ *Snack* are trivial MVDs in Movie_1 and Movie_2, respectively, and, therefore, decomposition (2.1) is equivalent to

$$\{(\mathit{Movie}_1(\mathit{Theater}, \mathit{Title}), \emptyset),\ (\mathit{Movie}_2(\mathit{Theater}, \mathit{Snack}), \emptyset)\}. \tag{2.2}$$

Thus, if we use a definition of dependency preservation similar to the definition for functional dependencies, we would say that (2.2) is not a dependency preserving decomposition since *Theater* $\rightarrow\!\!\rightarrow$ *Title* is not equivalent to the empty set of dependencies, if both are considered as sets of constraints over *Movie*(*Theater*, *Title*, *Snack*). Indeed, if a relation schema $(R[U], \Sigma)$ contains only multivalued dependencies, then all its 4NF decompositions are of the form $\{(R_i[U_i], \emptyset) \mid i \in [1, n]\}$, and no decomposition is dependency preserving under the definition for functional dependencies. The problem with this definition is that it does take into account all the semantic information available in the decomposition; if $\{(R_i[U_i], \emptyset) \mid i \in [1, n]\}$ is a lossless decomposition of $(R[U], \Sigma)$, then $\Sigma \models \bowtie[U_1, \ldots, U_n]$. We adopt here the definition of dependency preservation given by Yuan and Ozsoyoglu [YÖ86]: $\{(R_i[U_i], \emptyset) \mid i \in [1, n]\}$ is a dependency preserving decomposition of $(R[U], \Sigma)$ if $\Sigma \equiv \bowtie[U_1, \ldots, U_n]$, that is, for every instance I of $R[U]$, $I \models \Sigma$ if and only if $I \models \bowtie[U_1, \ldots, U_n]$. For example, (2.2) is a dependency preserving decomposition of (*Movie*(*Theater*, *Title*, *Snack*), $\{$*Theater* $\rightarrow\!\!\rightarrow$ *Title*$\}$) since *Theater* $\rightarrow\!\!\rightarrow$ *Title* is equivalent to $\bowtie[\{\mathit{Theater}, \mathit{Title}\}, \{\mathit{Theater}, \mathit{Snack}\}]$. Moreover, if Σ is a set of FDs and MVDs, then $\{(R_i[U_i], \Sigma_i) \mid i \in [1, n]\}$ is a dependency preserving decomposition of Σ if

$$\Sigma \;\equiv\; \{\bowtie[U_1, \ldots, U_n]\} \cup \bigcup_{i \in [1,n]} \Sigma_i.$$

Now, we turn our attention to the problem of 4NF decomposition. As in the case of functional dependencies, every relation schema admits a lossless 4NF decomposition, and in some cases a relation schema does not admit a dependency preserving 4NF decomposition. For instance, the relation schema (*Code*(*Address*, *City*, *PostalCode*), $\{$*PostalCode* $\rightarrow$ *City*, $\{$*Address*, *City*$\}$ $\rightarrow$ *PostalCode*$\}$), introduced in the previous section, does not admit a dependency preserving 4NF decomposition since $\{$*PostalCode* $\rightarrow$ *City*, $\{$*Address*, *City*$\}$ $\rightarrow$ *PostalCode*$\}$ is not equivalent to $\{$*PostalCode* $\rightarrow$ *City*, $\bowtie[\{\mathit{PostalCode}, \mathit{City}\}, \{\mathit{PostalCode}, \mathit{Address}\}]\}$.

In order to present a 4NF decomposition algorithm, we separately consider databases containing only multivalued dependencies and databases containing both functional and multivalued dependencies. Let $R[U]$ be relation schema and Σ a set of MVDs. Then, a

set $\Gamma := \Sigma$ and $Att := \{U\}$.
while there is $V \in Att$ and $X \twoheadrightarrow Y \in \Gamma$ such that $X \twoheadrightarrow Y$ is a nontrivial MVD in V
  $L := \{Y \mid Y \in dep(X) \text{ and } Y \cap V \neq \emptyset\}$
  for every $Y \in L$, $Z_1 \twoheadrightarrow Z_2 \in \Gamma$ and $W_1 \in LHS(\Gamma)$ do
    if $Z_1 \subseteq X(Y \cap V) \subseteq Z_1 Z_2$ and $Z_1 \twoheadrightarrow Z_2$ splits W_1
      then $\Gamma := \Gamma \cup \{Z_1(Z_2 \cap W_1) \twoheadrightarrow W_2 \mid W_2 \in dep(Z_1(Z_2 \cap W_1))\}$
  $Att := (Att - \{V\}) \cup \{X(Y \cap V) \mid Y \in L\}$
Output $\{(R_i[U_i], \emptyset) \mid i \in [1, n]\}$, where $Att = \{U_1, \ldots, U_n\}$.

Figure 2.13: A 4NF decomposition algorithm.

simple modification of the BCNF decomposition algorithm shown in Figure 2.11 leads to an algorithm for 4NF decomposition. If $(R[U], \Sigma)$ contains a nontrivial multivalued dependency $X \twoheadrightarrow Y$, then $(R[U], \Sigma)$ is split into two schemas $(R_1[XY], \pi_{XY}(\Sigma))$ and $(R_2[XZ], \pi_{XZ}(\Sigma))$, where $Z = U - XY$ and $\pi_V(\Sigma) = \{W_1 \twoheadrightarrow W_2 \cap V \mid \Sigma \models W_1 \twoheadrightarrow W_2 \text{ and } W_1 \subseteq V\}$. This process is repeated until no subschema contains a nontrivial multivalued dependency.

The previous algorithm has the same drawbacks of the BCNF decomposition algorithm shown in the previous section. Interestingly, Grahne and Räihä [GR83] developed a more efficient algorithm which materializes a suitable subset of $\pi_V(\Sigma)$. This algorithm works properly for any kind of multivalued dependencies, and it works in polynomial time if a simple condition is satisfied.

In order to present Grahne and Räihä's algorithm, we need to introduce some terminology. Let $(R[U], \Sigma)$ as in the previous paragraphs. Given $V \subseteq U$, we say that $X \twoheadrightarrow Y$ is a *nontrivial MVD in* V if $X \subsetneq X(Y \cap V) \subsetneq V$. Moreover, we say that $X \twoheadrightarrow Y$ *splits* a set of attributes Z if $(Y - X) \cap Z \neq \emptyset$ and $(U - XY) \cap Z \neq \emptyset$. Finally, $LHS(\Sigma)$ stands for the set of left hand sides in Σ. Grahne and Räihä's algorithm [GR83] is shown in Figure 2.13.

To understand the idea behind the algorithm shown in Figure 2.13, we review the first steps of this algorithm. Let $\Gamma = \Sigma$ and $Att = \{U\}$. If $(R[U], \Gamma)$ is not 4NF, then Σ contains a nontrivial multivalued dependency $X \twoheadrightarrow Y$. Thus, U is split by using the minimal sets of attributes implied by X, that is, $Att = \{XY \mid Y \in dep(X)\}$. Let XY be an element of Att. If XY is not in 4NF, then XY is split by using the same method. But how can we check whether XY is not in 4NF? A sufficient, but not necessary, condition is that there exists $Z \twoheadrightarrow W \in \Sigma$ such that $Z \twoheadrightarrow W$ is a nontrivial multivalued dependency in XY. Grahne and Räihä [GR83] proposed to add some dependencies to Σ to ensure that this is also a necessary condition. They showed that if all the consequences of the elements of $\mathit{LHS}(\Sigma)$ that are split by Y are included in Γ, then the previous condition is also necessary. The inner loop adds to Γ all these elements by using the following rule. If $Z_1 \twoheadrightarrow Z_2$ becomes a trivial MVD in XY ($Z_1 \subseteq XY \subseteq Z_1Z_2$) and Z_2 splits an element $W_1 \in \mathit{LHS}(\Sigma)$, then all the consequence of $Z_1(Z_2 \cap W_1)$ are added to this set. This process is repeated until all the subschemas are in 4NF.

The algorithm shown in Figure 2.13 works in polynomial time if no dependency in Σ splits an element $X \in \mathit{LHS}(\Sigma)$, since in this case no additional elements are added to Γ. It turns out that this class of multivalued dependencies properly contains the class of *conflict-free* multivalued dependencies, which is defined as follows. A set of MVD Σ is conflict-free if no dependency in Σ splits an element $X \in \mathit{LHS}(\Sigma)$ and for every $X, Y \in \mathit{LHS}(\Sigma)$, $\mathit{dep}(X) \cap \mathit{dep}(Y) \subseteq \mathit{dep}(X \cap Y)$. Conflict-free MVDs were widely studied in the database literature [Sci81, FMU82, BFMY83, YÖ86] and they were claimed to be the most "natural" class of multivalued dependencies [Sci81].

Finally, we consider the problem of normalizing a database containing functional and multivalued dependencies. Let $R[U]$ be a relation schema, Σ a set of FDs over $R[U]$ and Γ a set of MVDs over $R[U]$. Most database textbooks do not pay much attention to the 4NF decomposition problem in the presence of functional and multivalued dependencies [EN99, GMUW01]. To solve this problem, they simply propose to transform Σ into a set of multivalued dependencies $\overline{\Sigma} = \{X \twoheadrightarrow A \mid X \rightarrow Y \in \Sigma \text{ and } A \in Y\}$ and then use a 4NF decomposition algorithm for multivalued dependencies. It was shown by Yuan and Ozsoyoglu [YÖ86] that this is not a good approach; it could be the case that $(R[U], \Sigma \cup \Gamma)$ admits a lossless and dependency preserving 4NF decomposition, but it cannot be obtained by normalizing $(R[U], \overline{\Sigma} \cup \Gamma)$. The problem is that the different semantics of FDs and MVDs are neglected. To overcome this limitation, Yuan and Ozsoyoglu [YÖ86] propose to apply a 4NF decomposition algorithm for multivalued dependencies to the following transformation of $\Sigma \cup \Gamma$:

$$\begin{aligned} \mathit{Envelope}(\Sigma \cup \Gamma) \ = \ \{X \twoheadrightarrow Y \mid X \in \mathit{LHS}(\Sigma \cup \Gamma),& \\ \Sigma \cup \Gamma \models X \twoheadrightarrow Y \text{ and } \Sigma \cup \Gamma \not\models X \rightarrow Y\}.& \end{aligned}$$

In [YÖ86], it was proved that a 4NF decomposition of $(R[U], \mathit{Envelope}(\Sigma \cup \Gamma))$ is a

4NF decomposition of $(R[U], \Sigma \cup \Gamma)$. Moreover, Yuan and Ozsoyoglu showed that if *Envelope*$(\Sigma \cup \Gamma)$ is conflict-free, then $(R[U], \Sigma \cup \Gamma)$ has a lossless and dependency preserving 4NF decomposition.

Projection/Join Normal Form (PJ/NF)

In the previous sections, we introduce some normal forms for functional dependencies and multivalued dependencies. The next natural step is to define a normal form for join dependencies. It turns out that defining such a normal form is more complicated than in the previous cases. To see why, consider the most natural extension of 4NF to join dependencies. Let $R[U]$ be a relation schema and Σ a set of FDs and JDs. Then, $(R[U], \Sigma)$ is in fifth normal form (5NF) if for every nontrivial join dependency $\bowtie[X_1, \ldots, X_n]$ implied by Σ, X_i $(1 \leq i \leq n)$ is a superkey. So, 5NF is a simple generalization of 4NF. If $X \rightarrow\!\!\!\rightarrow Y$ is a nontrivial multivalued dependency implied by Σ, then $\bowtie[XY, XZ]$ is implied by Σ, where $Z = U - XY$. Thus, if $(R[U], \Sigma)$ is in 5NF, then XY and XZ are superkeys and, therefore, X is a superkey, since $\{\bowtie[XY, XZ],\ XY \rightarrow U,\ XZ \rightarrow U\} \models X \rightarrow U$. Hence, $(R[U], \Sigma)$ is in 4NF.

This normal form is a very stringent requirement, as pointed out by Vincent [Vin97]. If a join dependency $\bowtie[X_1, \ldots, X_n]$ is nontrivial, then $\bowtie[X_1, \ldots, X_n, A]$ is also nontrivial, for every attribute A. Thus, if $(R[U], \Sigma)$ is in 5NF, then every attribute must be a superkey. This condition is virtually unattainable in practice.

A first definition of a normal form for join dependencies was provided by Fagin [Fag79]. Let $(R[U], \Sigma)$ be as above and $KD(\Sigma) = \{X \rightarrow U \mid X \subseteq U \text{ and } \Sigma \models X \rightarrow U\}$. Notice that $\Sigma \models KD(\Sigma)$, but $KD(\Sigma)$ does not necessarily implies Σ. Fagin [Fag79] observed that if Σ contains only functional and multivalued dependencies, then Σ is in 4NF if and only if $KD(\Sigma) \models \Sigma$. In particular, if Σ contains only functional dependencies, then $(R[U], \Sigma)$ is in BCNF if and only if $KD(\Sigma) \models \Sigma$. Fagin considered these properties to generalize BCNF and 4NF to the case of join dependencies. If Σ contains functional dependencies and join dependencies, then $(R[U], \Sigma)$ is in projection-join normal form (PJ/NF) if $KD(\Sigma) \models \Sigma$. Moreover, a database schema S is in PJ/NF if every relation schema in S is in PJ/NF.

In this section, we will only focus on the problem of testing whether a relation schema is in PJ/NF, and we do not elaborate on the question of how to decompose a relation schema into a PJ/NF database schema. The latter problem can be solved by using a decomposition algorithm similar to the decomposition algorithms for functional dependencies and multivalued dependencies.

The problem of testing whether a relation schema is in PJ/NF can be solved by using a simple algorithm. For every set of attributes $X \subseteq U$, use Maier et al. [MSY81] algorithm (see Section 2.1.2) to test in polynomial time whether X is a superkey. Then, check if $KD(\Sigma) \models \Sigma$ by using the chase (see Section 2.1.2). This algorithm requires exponential time. To the best of our knowledge, the exact complexity of the PJ/NF testing problem remains open. Interestingly, Date and Fagin [DF92] proposed a sufficient condition that ensures that a schema is in PJ/NF and it can be tested in polynomial time. Let $R[U]$ be a relation schema and Σ a set of FDs and JDs. Then, $(R[U], \Sigma)$ is in BCNF [DF92] if for every nontrivial FD $X \rightarrow A \in \Sigma^+$, X is a superkey. We note that this definition extends the definition of BCNF to the case of functional and join dependencies. Furthermore, a key in $(R[U], \Sigma)$ is simple if it consists of a single attribute. Date and Fagin [DF92] proved that if $(R[U], \Sigma)$ is in BCNF and every key in $(R[U], \Sigma)$ is simple, then $(R[U], \Sigma)$ is in PJ/NF [8].

We end this section by showing how to test in polynomial time whether a relation schema $(R[U], \Sigma)$ contains only simple keys and is in BCNF. Let $N = \{A \mid A \in U \text{ and } \Sigma \not\models A \rightarrow U\}$. Then, $(R[U], \Sigma)$ contains only simple keys if and only if $\Sigma \not\models N \rightarrow U$. Thus, by using the FD implication algorithm presented in Section 2.1.2 we can test in polynomial time whether $(R[U], \Sigma)$ contains only simple keys. We note that since Σ contains join dependencies, we cannot test whether $(R[U], \Sigma)$ is in BCNF by checking that for every $X \rightarrow Y \in \Sigma$, X is a superkey. For example, if $U = ABCDE$ and $\Sigma = \{E \rightarrow ABCD, \bowtie[AB, CD, E]\}$, then $(R[U], \Sigma)$ is not in BCNF ($\Sigma \models AB \rightarrow CD$ and AB is not a superkey) but $E \rightarrow ABCD$ is the only functional dependency in Σ and E is a superkey. We verify whether $(R[U], \Sigma)$ is in BCNF as follows. Assume that $(R[U], \Sigma)$ contains only simple keys. Then, $(R[U], \Sigma)$ is in BCNF if and only if for every $A \in N$, $\Sigma \not\models N - \{A\} \rightarrow A$. This condition can be checked in polynomial time by using the FD implication algorithm presented in Section 2.1.2.

Reduced-Fifth Normal Form (5NFR)

Vincent [Vin97] showed that PJ/NF can be too restrictive. For example, a database schema containing functional dependencies $AB \rightarrow C$, $AC \rightarrow B$, $BC \rightarrow A$ and a join

[8] For a discussion on the usefulness of this condition see [Buf93, DF93].

dependency $\bowtie[AB, AC, BC]$ is not in PJ/NF since $\{AB \rightarrow C,\ AC \rightarrow B,\ BC \rightarrow A\} \not\models \bowtie[AB, AC, BC]$. But this specification is not prone to update anomalies, since if a tuple t is in the join of tuples t_1, t_2, t_3, then t is equal to either t_1 or t_2 or t_3.

Vincent [Vin97] proposed a less restrictive normal form for functional and join dependencies. Let $R[U]$ be a relation schema and Σ a set of FDs and JDs. A join dependency $\bowtie[X_1, \ldots, X_n] \in \Sigma$ is *strong-reduced* if for every $i \in [1, n]$, $\Sigma \not\models \bowtie[X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n]$ or $X_1 \cup \cdots \cup X_{i-1} \cup X_{i+1} \cup \cdots \cup X_n \subsetneq U$. Then, $(R[U], \Sigma)$ is in reduced-fifth normal form (5NFR) if for every nontrivial, strong-reduced join dependency $\bowtie[X_1, \ldots, X_n] \in \Sigma^+$ and every $i \in [1, n]$, X_i is a superkey. Vincent [Vin97] proved that if a database schema is in PJ/NF, then it is in 5NFR. Furthermore, Vincent showed that if Σ contains only FDs and strong-reduced JDs, then $(R[U], \Sigma)$ is in 5NFR if and only if for every $X \rightarrow Y \in \Sigma$, X is a superkey, and for every $\bowtie[X_1, \ldots, X_n] \in \Sigma$ and every $i \in [1, n]$, X_i is a superkey. Thus, the relation schema shown above is in 5NFR ($\bowtie[AB, AC, BC]$ is strong-reduced and AB, AC and BC are superkeys) and, therefore, PJ/NF is strictly stronger than 5NFR.

Domain Key Normal Form (DK/NF)

In general, a data dependency over a relation schema $R[U]$ is a mapping f from $\{I \mid I$ is an instance of $R[U]\}$ to $\{\mathit{true}, \mathit{false}\}$; it defines the set of valid instances of $R[U]$, that is, $\{I \mid I$ is an instance of $R[U]$ and $f(I) = \mathit{true}\}$. Fagin [Fag81] proposed the ultimate normal form for any type of data dependencies. Let Σ be a set of data dependencies over $R[U]$, $KD(\Sigma)$ the set of key dependencies implied by Σ and $DD(\Sigma)$ the set of domain dependencies implied by Σ. Recall that a domain dependency defines the domain of an attribute and is an expression of the form $IN(A, D)$, where $A \in U$ and $D \subseteq Dom(A)$. Then, $(R[U], \Sigma)$ is in domain key normal form (DK/NF) if $KD(\Sigma) \cup DD(\Sigma) \models \Sigma$ [Fag81].

In general, the problem of DK/NF testing is undecidable, and it becomes decidable if we consider a class of dependencies for which the implication problem is decidable, such as FDs and JDs. In the rest of this section, we present a simple fragment of first order logic for which this problem is decidable. This fragment is interesting since it provides a uniform way of representing functional dependencies, join dependencies and some domain dependencies (if constants are allowed) as well as other data dependencies.

A *universal* sentence over a relation schema $R[U]$ is a first order sentence of the form $\forall \bar{x} \psi$, where ψ is a quantifier-free formula. Functional dependencies and join dependencies can be expressed by using universal sentences. For example, a functional dependency $A \rightarrow B$ in a relation schema R with attributes A, B, C can be represented as $\forall x \forall y_1 \forall z_1 \forall y_2 \forall z_2\, (R(x, y_1, z_1) \wedge R(x, y_2, z_2) \rightarrow y_1 = y_2)$, while a join dependency $\bowtie[AB, AC, BC]$ can be represented as

$$\forall x \forall y \forall z \forall u_1 \forall u_2 \forall u_3\, (R(x, y, u_1) \wedge R(x, u_2, z) \wedge R(u_3, y, z) \rightarrow R(x, y, z)).$$

Some domain dependencies can also be expressed by using universal sentences. For example, a domain dependency $IN(A, \{1, 2, 3\})$ can be represented as $\forall x \forall y \forall z (R(x, y, z) \rightarrow (x = 1 \vee x = 2 \vee x = 3))$.

The implication problem for universal sentences can be reduced to the problem of testing if a formula of the form $\exists \bar{x} \forall \bar{y} \psi$ is satisfiable, where ψ is a quantifier-free formula. This is a Schönfinkel-Bernays expression and it can be tested in nondeterministic exponential time whether this sentence is satisfiable [AHV95]. By using this result, it is possible to construct a simple algorithm for testing whether a database schema containing universal sentences is in DK/NF: materialize $KD(\Sigma) \cup DD(\Sigma)$ and verify whether $KD(\Sigma) \cup DD(\Sigma) \models \Sigma$. This algorithm runs in double exponential time.

2.1.4 Why are Normalized Databases Good?

Even though normalization theory is one of the most thoroughly researched subjects in database theory (see [BBG78] for an early survey on normalization, and see [Kan90] for a more recent survey), the problem of *formally proving* that normal forms are *good* has not received much attention in the database literature. In this section, we summarize the work that has been done in this area. First, we present two different approaches for characterizing insertion and deletion anomalies, and we show that in both approaches BCNF is precisely the normal form that guarantees no anomalies, if only functional dependencies are provided. In this section, we also consider 4NF and DK/NF. Second, we characterize some normal forms in terms of their ability to eliminate redundant information.

Insertion and Deletion Anomalies

Probably the first attempt to prove that BCNF eliminates insertion and deletion anomalies is due to Bernstein and Goodman [BG80]. In this paper, they formalized the notion of insertion anomaly in terms of functional dependencies that are *affected* when a new tuple is inserted. For example, Figure 2.14 shows a database instance of relation

ISBN	*Title*	*Author*
155860622X	Data on the Web	Serge Abiteboul
155860622X	Data on the Web	Dan Suciu

Figure 2.14: An instance of the relation schema (*Book*(*ISBN*, *Title*, *Author*), $\{$*ISBN* $\rightarrow$ *Title*$\}$).

schema (*Book*(*ISBN*, *Title*, *Author*), $\{$*ISBN* $\rightarrow$ *Title*$\}$) for storing information about books. Observe that this schema is not in BCNF. A tuple (0201537710, Foundations of Databases, Victor Vianu) affects FD *ISBN* $\rightarrow$ *Title* since new values are inserted into the first and second columns of the instance and, therefore, they can produce a violation of FD *ISBN* $\rightarrow$ *Title*. On the other hand, tuple (155860622X, Data on the Web, Peter Buneman) does not affect this functional dependency. This schema is undesirable since the effects of an insertion cannot be predicted simply by examining the schema: some insertions affect *ISBN* $\rightarrow$ *Title* while others do not affect this functional dependency. Intuitively, a relation schema is free of insertion anomalies if it is *syntactically predictable* [BG80], that is, the effects of an insertion can be determined by checking the schema alone.

Formally, let $S = (R[U], \Sigma)$ be a relation schema containing only functional dependencies, I a database instance of S and t a U-tuple. Then, $I \cup \{t\}$ *affects* a functional dependency $X \rightarrow Y$ if $\pi_{XY}(I) \subsetneq \pi_{XY}(I \cup \{t\})$. Furthermore, *Affect*(S) is the set of all nontrivial functional dependencies φ in Σ^+ that are affected by some $I \cup \{t\}$, where I is a nonempty database instance I of S and t is a U-tuple, and $\mathit{NoAffect}(S)$ is the set of all nontrivial functional dependencies φ in Σ^+ such that there exists a nonempty database instance I of S and a U-tuple $t \notin I$ such that $I \cup \{t\}$ does not affect φ. In the example shown above:

$$\begin{aligned} \mathit{Affect}(S) &= \{\mathit{ISBN} \rightarrow \mathit{Title},\ \{\mathit{ISBN}, \mathit{Author}\} \rightarrow \{\mathit{ISBN}, \mathit{Title}, \mathit{Author}\}\},\\ \mathit{NoAffect}(S) &= \{\mathit{ISBN} \rightarrow \mathit{Title}\}. \end{aligned}$$

Notice that a key is always affected by the insertion of a new tuple and, therefore, it cannot be in *NoAffect*(S). Bernstein and Goodman [BG80] defined a relation schema S as *free of insertion anomalies* if $\mathit{Affect}(S) \cap \mathit{NoAffect}(S) = \emptyset$, and they proved that S is free of insertion anomalies if and only if S is in BCNF. Bernstein and Goodman also characterized deletion anomalies in terms of functional dependencies that are affected when a tuple is removed from the database, and they proved that a relation schema S is free of deletion anomalies if and only if S is in BCNF [BG80].

The problem of characterizing BCNF in terms of insertion and deletion anomalies was also considered by LeDoux and Parker [LJ82], but their results are weaker than Bernstein and Goodman's characterization of BCNF [BG80]. The problem of characterizing DK/NF (see Section 2.1.3), and in particular BCNF and 4NF, in terms of insertion and deletion anomalies was considered by Fagin [Fag81]. In this paper, Fagin introduced the notions of *key-based* insertion and deletion anomalies. Let $S = (R[U], \Sigma)$ be a relation schema, I a database instance of S and t a U-tuple not in I. Then t is *compatible with* I if (1) for every domain dependency $IN(A, D) \in \Sigma$, $t[A] \in D$, and (2) for every key dependency $X \to U \in \Sigma^+$ and every $s \in I$, $t[X] \neq s[X]$. For example, tuple (155860622X, Foundations of Databases, Peter Buneman) is compatible with the database instance shown in Figure 2.14. Relation schema S has a *key-based insertion anomaly* if there exists an instance I of S and a U-tuple t compatible with I such that $I \cup \{t\}$ does not satisfy Σ. Moreover, S has a *key-based deletion anomaly* if there exists an instance I of S and $t \in I$ such that $I - \{t\}$ does not satisfy Σ. For example, the schema of the instance I shown in Figure 2.14 does not have a key-based deletion anomaly and it has a key-based insertion anomaly since tuple $t =$ (155860622X, Foundations of Databases, Peter Buneman) is compatible with I and $I \cup \{t\} \not\models \mathit{ISBN} \to \mathit{Title}$.

Fagin [Fag81] showed that a database schema S is free of key-based insertion and deletion anomalies if and only if S is in DK/NF. In particular, if the schema contains only functional and multivalued dependencies, then S is free of key-based insertion and deletion anomalies if and only if S is in 4NF. The same corollary is obtained for the case of BCNF and functional dependencies alone.

Redundant Information

A goal of normalization theory is to eliminate redundant information. Vincent [Vin99] formalized the notion of redundancy for database schemas containing functional and multivalued dependencies and proved that 4NF, and in particular BCNF, eliminates redundant information. More precisely, let $S = (R[U], \Sigma)$ be a relation schema containing functional and multivalued dependencies, I a database instance of S and $t \in I$. Then, a value $t[A]$ is *redundant in* I [Vin99], for some $A \in U$, if for every $a \neq t[A]$, the instance I' obtained by replacing $t[A]$ by a does not satisfy Σ. For example, the value "Data on

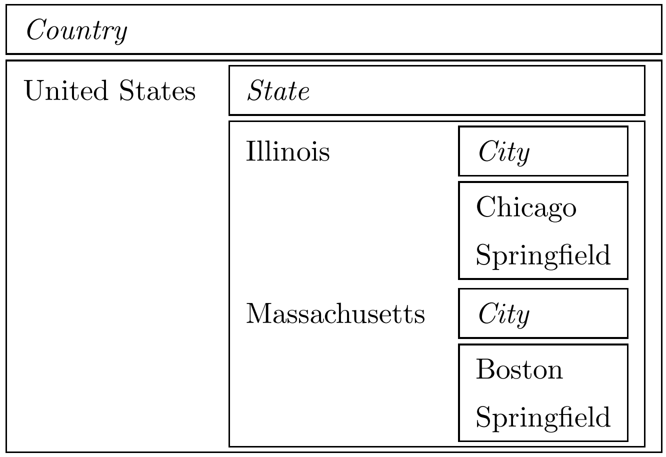


Figure 2.15: A nested relation.

the Web" in the first row of the database instance shown in Figure 2.14 is redundant, since any replacement of this value by a new one leads to an instance which does not satisfy $ISBN \rightarrow Title$. Vincent [Vin99] defined a relation schema S as redundant if there exists an instance I of S containing a redundant value. Moreover, Vincent showed that a relation schema S containing FDs and MVDs is not redundant if and only if S is in 4NF. A corollary of this theorem is that if S contains only functional dependencies, then S is not redundant if and only if S is in BCNF.

2.2 Nested Relational Databases

The basic assumption in the relational model is that every tuple in a relation contains atomic values. In some applications such as text processing, picture data processing and computer aided design [Mak77, ÖY87] this assumption is not appropriate; these applications require relations whose tuple components are sets or even relations themselves. To overcome this limitation, Makinouchi [Mak77] introduced the nested relational model.

In the nested relational model, a nested relation is a finite set of tuples whose components are atomic values or nested relations. For example, Figure 2.15 shows a binary nested relation containing one tuple. The first component of this tuple is an atomic value for the attribute *Country*, and its second component is a nested relation containing two tuples $(\text{Illinois}, \{\text{Chicago}, \text{Springfield}\})$, $(\text{Massachusetts}, \{\text{Boston}, \text{Springfield}\})$.

Every nested relation has associated a nested relation schema. A nested relation schema [Mak77, ÖY87] is either a set of attributes X, or $X(S_1)^* \ldots (S_n)^*$, where S_i

$(i \in [1, n])$ is a nested relation schema. For example, the nested relation schema of the relation shown in Figure 2.15 is $S_1 = \mathit{Country}(S_2)^*$, $S_2 = \mathit{State}(S_3)^*$, $S_3 = \mathit{City}$. This relation contains one tuple, say t, such that $t[\mathit{Country}]$ is an atomic value and $t[S_2]$ is a nested relation of the nested relation schema S_2. Moreover, a nested database schema is a set of nested relation schemas. Usually, when defining a nested schema we omit the names of the nested subschemas. Thus, $S_1 = \mathit{Country}(\mathit{State}(\mathit{City})^*)^*$ is the nested relation schema of the relation shown in Figure 2.15.

In the following sections, we will introduce data dependencies and normalization theory for nested relational databases. In these sections, the following concepts will play a central role. Let S be a nested relation schema. If $S = X$, where X is a set of attributes, then $\mathit{Attribute}(S) = X$. Otherwise, if $S = X(S_1)^* \ldots (S_n)^*$, then $\mathit{Attribute}(S) = X \cup \mathit{Attribute}(S_1) \cup \cdots \cup \mathit{Attribute}(S_n)$. Given a nested relation I of S, the *total unnesting* of I, denoted by $TU(I)$, is recursively defined as follows. If $S = X$, where X is a set of attributes, then $TU(I) = I$. If S is of the form $X(S_1)^* \cdots (S_n)^*$ and $X_i = \mathit{Attribute}(S_i)$ $(i \in [1, n])$, then

$$
\begin{aligned}
TU(I) \quad = \quad & \{t \mid t \text{ is a } \mathit{Attribute}(\mathrm{S})\text{-tuple and there exists a tuple } u \text{ in } I \text{ such that} \\
& \quad t[X] = u[X] \text{ and } t[X_i] \text{ is a tuple in the total unnesting of } u[S_i]\}.
\end{aligned}
$$

For example, the total unnesting of the nested relation shown in Figure 2.15 is shown in Figure 2.16.

The total unnesting of a nested relation I is a flat representation of this relation. It is possible to reconstruct I from this flat representation if I is in *partition normal form* [RKS88]. Formally, if I is a nested relation of a schema $X(S_1)^* \ldots (S_n)^*$, then I is in *partition normal form* (PNF) if for any tuple t in I: (1) there is no t' in I such that $t[X] = t'[X]$ and $t \neq t'$; and (2) each nested relation $t[S_i]$ $(i \in [1, n])$ is in PNF. For example, the nested relation shown in Figure 2.15 is in PNF. From now on, as it is usually done in nested relational databases [RKS88, ÖY87, MNE96], we assume that every nested relation is in PNF.

2.2.1 Data Dependencies in Nested Relational Databases

Two main approaches have been followed in order to define data dependencies for nested relational databases. In the first approach (flat approach), data dependencies are defined in terms of tuples in the total unnesting of a nested relation [ÖY87, ÖY89, MNE96,

Country	*State*	*City*
United States	Illinois	Chicago
United States	Illinois	Springfield
United States	Massachusetts	Boston
United States	Massachusetts	Springfield

Figure 2.16: Total unnesting of nested relation shown in Figure 2.15.

Mok02], while in the second one (nested approach) data dependencies are directly defined in terms of the tuples in a nested relation or the values that can be reached by traversing them [Mak77, FSTG85, HD99]. To the best of our knowledge, in both approaches only functional dependencies and multivalued dependencies have been considered. We present both approaches next.

The Nested Approach

Functional dependencies and multivalued dependencies can be easily generalized to the case of nested relations. Let I be a nested relation of a nested schema $X(S_1)^* \dots (S_n)^*$ and Y, Z nonempty subsets of $X \cup \{S_1, \dots, S_n\}$. Then, I satisfies a functional dependency $Y \rightarrow Z$, denoted by $I \models Y \rightarrow Z$, if for every $t_1, t_2 \in I$, $t_1[Z] = t_2[Z]$ whenever $t_1[Y] = t_2[Y]$ [Mak77, FSTG85]. Furthermore, I satisfies a multivalued dependency $Y \rightarrow\!\!\!\rightarrow Z$, denoted by $I \models Y \rightarrow\!\!\!\rightarrow Z$, if for every $t_1, t_2 \in I$ such that $t_1[Y] = t_2[Y]$, there exists $t_3 \in I$ such that $t_3[YZ] = t_1[YZ]$ and $t_3[YW] = t_2[YW]$, where $W = U - YZ$ [Mak77, FSTG85]. Observe that in these definitions we are considering set theoretic equality. For example, Figure 2.15 shows a nested relation of nested schema $S_1 = \mathit{Country}(S_2)^*$, $S_2 = \mathit{State}(S_3)^*$, $S_3 = \mathit{City}$. This relation satisfies FD $\mathit{Country} \rightarrow S_2$ [9].

This simple approach was followed by Makinouchi [Mak77] to introduce functional dependencies for nested relations. Fischer et al. [FSTG85] study the relationship between functional dependencies and multivalued dependencies in a nested relation and in its total unnesting. For instance, the schema of the total unnesting of the nested relation I shown in Figure 2.15 contains attributes *Country*, *State*, *City*, and FD $\mathit{Country} \rightarrow S_2$ corresponds to FD $\mathit{Country} \rightarrow \{\mathit{State}, \mathit{City}\}$ in this schema. This functional dependency does not hold in the total unnesting of I (see Figure 2.16).

[9]Indeed, any relation I of nested schema S_1 satisfies FD $\mathit{Country} \rightarrow S_2$ since I is in PNF.

Makinouchi [Mak77] considers only nested schemas with one level of nesting, that is, schemas of the form $X(X_1)^* \ldots (X_n)^*$, where $X, X_1, \ldots, X_n$ are sets of attributes. The main drawback of his definition of functional dependency is that it cannot be directly extended to nested schemas with a larger degree of nesting. For instance, what is the meaning of the FD *City* $\rightarrow$ *State* in the nested relation shown in Figure 2.15? Intuitively, I does not satisfy this constraint since two distinct states have Springfield as a city. Formally, we need to check whether for every $t_1, t_2 \in I$, if $t_1[\mathit{City}] = t_2[\mathit{City}]$, then $t_1[\mathit{State}] = t_2[\mathit{State}]$. But, for a tuple t in I, what are the values of $t[\mathit{City}]$ and $t[\mathit{State}]$? If we assume that $t[\mathit{City}]$ is the set of all values mentioned in the column *City* for this tuple, and likewise for $t[\mathit{State}]$, then $I \models \mathit{City} \rightarrow \mathit{State}$ since for each $t_1, t_2 \in I$, $t_1[\mathit{City}] = t_2[\mathit{City}] = \{$Chicago, Springfield, Boston$\}$ and $t_1[\mathit{State}] = t_2[\mathit{State}] = \{$Illinois, Massachusetts$\}$.

To overcome this limitation, Hara and Davidson [HD99] considered functional dependencies defined in terms of paths of attributes (this approach was previously followed by Weddell [Wed92] to introduce functional dependencies for an object-oriented data model). Formally, given a nested relation schema $S = X(S_1)^* \ldots (S_n)^*$, p is a path on S if $p = \epsilon$ (empty path) or p is of the form $A.p'$, where $A \in X$ and there exists $i \in [1, n]$ such that p' is path on S_i. For example, *Country.State* and *Country.State.City* are paths on the nested schema considered above. Then, a functional dependency over S is an expression of the form [HD99]:

$$p_0\,[\,p_1, \ldots, p_n \rightarrow p_{n+1}\,], \tag{2.3}$$

where p_0, $p_0.p_1$, ..., $p_0.p_n$, $p_0.p_{n+1}$ are paths on S. For instance,

$$\mathit{Country}\,[\,\mathit{State.City} \rightarrow \mathit{State}\,], \tag{2.4}$$

is a functional dependency over the nested schema shown above. We use this FD to present Hara and Davidson's semantics for functional dependencies [HD99]. We do not formally present this semantics since, to the best of our knowledge, all the normal forms and normalization algorithms for nested relational databases presented in the literature consider only data dependencies defined by using the flat approach (see next section).

Hara and Davidson's approach [HD99] implicitly assume that nested relations can be represented as trees. Figure 2.17 (a) shows a labeled tree T_I representing nested relation I shown in Figure 2.15, and Figure 2.17 (b) shows a labeled tree $T_{I'}$ representing a nested relation I' containing two tuples: $(\mathrm{C}_1, \{\mathit{Illinois}, \{\mathit{Chicago}, \mathit{Springfield}\}\})$ and

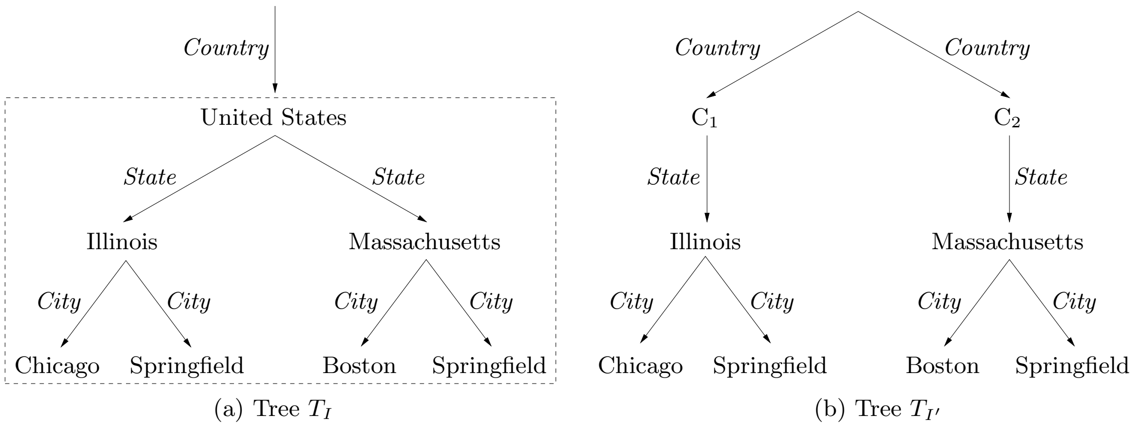


Figure 2.17: Labeled trees representing nested relations.

$(\mathrm{C}_2, \{\mathit{Massachusetts}, \{\mathit{Boston}, \mathit{Springfield}\}\})$. Notice that attributes are used as labels of the edges. By following a path in these trees we reach some values. For example, Illinois is reachable from the root of T_I by following path *Country.State*, so is Massachusetts.

The prefix path *Country* in FD (2.4) (in general, p_0 in FD (2.3)) indicates in which subtrees $\mathit{State.City} \rightarrow \mathit{State}$ is evaluated; this FD is evaluated in all the subtrees whose root is reachable by following path *Country*. One of these subtrees is shown in Figure 2.17 (a) inside a dotted rectangle. Assume that T is that subtree and let r be its root. Then, T satisfies $\mathit{State.City} \rightarrow \mathit{State}$ if for every v_1, v_2 reachable from r by following path *State.City*, if $v_1 = v_2$, then the intermediate values reached by following the prefix path *State* are equal. Thus, T does not satisfy $\mathit{State.City} \rightarrow \mathit{State}$ since by following path *State.City* we can reach value Springfield through two distinct intermediate values Illinois and Massachusetts. Observe that I' satisfies (2.4) since the subtrees rooted at C_1 and C_2 satisfy $\mathit{State.City} \rightarrow \mathit{State}$. Hence, (2.4) expresses the following functional dependency: In every country, cities in distinct states have different names. Notice that if (2.4) is replaced by $\epsilon\,[\,\mathit{Country.State.City} \rightarrow \mathit{Country.State}\,]$, then neither I nor I' satisfies the new functional dependency.

Finally, it is worth mentioning that Hara and Davidson [HD99] presented a sound and complete set of eight inference rules for a subclass of functional dependencies satisfying the following condition: empty sets cannot occur in any nested relation.

The Flat Approach

The flat approach has two main advantages. First, FDs and MVDs are simple to define. The total unnesting of a nested relation is a usual relation, and, therefore, functional dependencies and multivalued dependencies are defined as usual. Let S be a nested relation schema and $X, Y \subseteq \mathit{Attribute}(S)$. Then, a nested relation I of S satisfies $X \to Y$ ($X \to\!\!\to Y$) if $TU(I) \models X \to Y$ ($TU(I) \models X \to\!\!\to Y$) [ÖY87]. For example, functional dependency (2.4) can be represented as[10]:

$$\{\mathit{Country},\ \mathit{City}\} \to \mathit{State}.$$

Nested relation I shown in Figure 2.15 does not satisfy this constraint since its total unnesting, shown in Figure 2.16, does not satisfy this functional dependency in the usual sense (see Section 2.1.2).

The second advantage of the flat approach is that the implication problem for FDs and MVDs can be reduced to the same problem for relational databases. Thus, the efficient methods presented in Section 2.1.2 can be used for nested relational databases.

2.2.2 Nested Relational Databases Design

Consider again the relation schema $\mathit{Movie}(\mathit{Theater}, \mathit{Title}, \mathit{Snack})$ introduced in Section 2.1.2. Recall that a tuple (th, ti, sn) is in this database if theater th is showing movie ti and offering snack sn. Figure 2.18 (a) shows one instance of the relation *Movie*. For a given theater, the information about titles and snacks is independent and, therefore, this schema satisfies MVD $\mathit{Theater} \to\!\!\to \mathit{Title}$. Given that *Theater* is not a key, this database specification is not in 4NF and is prone to update anomalies. To solve this problem, we can split the database into two new relations; one containing information about theaters and titles, and the other one containing information about theaters and snacks.

As mentioned in [Fag77], the MVD $\mathit{Theater} \to\!\!\to \mathit{Title}$ implies that *Title* and *Snack* are "orthogonal" or "independent" columns names. This orthogonality holds in the sense that the information about a particular theater, say Bloor Cinema, can be represented as the cross product:

$$\{\text{Bloor Cinema}\} \times \{\text{Bad Company, Spider-Man}\} \times \{\text{coffee, popcorn}\}.$$

[10] There exists functional dependencies that can be expressed by using Hara and Davidson's language [HD99] and cannot be expressed by using the flat approach.

Theater	*Title*	*Snack*
Bloor Cinema	Bad Company	coffee
Bloor Cinema	Bad Company	popcorn
Bloor Cinema	Spider-Man	coffee
Bloor Cinema	Spider-Man	popcorn
Paramount	Bad Company	coke
Paramount	Bad Company	popcorn
Paramount	Insomnia	coke
Paramount	Insomnia	popcorn
Paramount	Spider-Man	coke
Paramount	Spider-Man	popcorn

(a) An instance of relation *Movie*

<table>
<tr><th colspan="3">Theater</th></tr>
<tr><td rowspan="2">Bloor Cinema</td><td>Title</td><td>Snack</td></tr>
<tr><td>Bad Company
Spider-Man</td><td>coffee
popcorn</td></tr>
<tr><td rowspan="2">Paramount</td><td>Title</td><td>Snack</td></tr>
<tr><td>Bad Company
Insomnia
Spider-Man</td><td>coke
popcorn</td></tr>
</table>

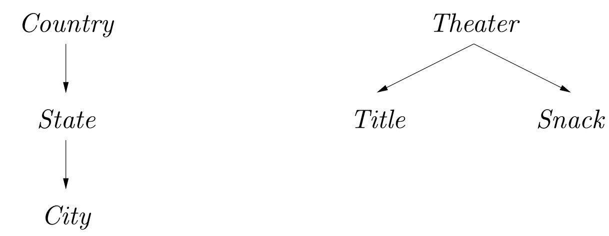
(b) A nested representation of relation *Movie*

Figure 2.18: *Movie* relation as a nested relation.

Therefore, to avoid update anomalies, instead of splitting the information about theaters into two relations, we can use a nested relation. For each theater, this relation stores a set of titles and a set of snacks. The schema of this database is $Theater(Title)^*(Snack)^*$. Figure 2.18 (b) shows a nested relation equivalent[11] to the instance of *Movie* shown in Figure 2.18 (a).

In general, given a relational database schema containing functional and multivalued dependencies, it is possible to use nested relations to "normalize" this schema. In this way, nested relations reduce redundancy and eliminate update anomalies. Based upon this idea, Özsoyoglu and Yuan [ÖY87, ÖY89] and Mok et al. [MNE96] introduced three normal forms for nested relational databases. Given a nested relation schema S such that $Attribute(S) = U$ and a set Σ of functional dependencies and multivalued dependencies over S, defined by using the flat approach, these normal forms define when S is a *good* grouping of the set of attributes U in the sense that: (1) redundant information and updates anomalies are eliminated and (2) a good representation of the semantic information contained in Σ is constructed. All these normal forms were called Nested Normal Form (NNF) by their authors. To distinguish between them we will use the following notation: NNF-87 for the normal form introduced in [ÖY87], NNF-89 for [ÖY89] and NNF-96 for

[11] The total unnesting of the nested relation shown in Figure 2.18 (b) is the relation shown in Figure 2.18 (a).



(a) $Country(State(City)^*)^*$ (b) $Theater(Title)^*(Snack)^*$

Figure 2.19: Schema trees of two nested schemas.

[MNE96]. We present them next.

NNF-87 and NNF-89

In order to present the nested normal forms introduced by Özsoyoglu and Yuan [ÖY87, ÖY89] we need to introduce some terminology. Given a nested relation schema S, the *schema tree* of S, denoted by $SchemaTree(S)$, is a tree defined as follows. If $S = X$, where X is a set of attributes, then $SchemaTree(S)$ contains only one node labeled X. If $S = X(S_1)^* \ldots (S_n)^*$, then the root of $SchemaTree(S)$ is a node labeled X and its children are the roots of $SchemaTree(S_1)$, ..., $SchemaTree(S_n)$. For example, the schema trees of nested relation schemas $Country(State(City)^*)^*$ and $Theater(Title)^*(Snack)^*$ are shown in Figures 2.19 (a) and (b), respectively. Given a node Y in $SchemaTree(S)$, $Ancestor(Y)$ is the union of labels of all ancestors of Y in this tree, including Y, and $Descendant(Y)$ is the union of labels of all descendants of Y in this tree, including Y. For example, $Ancestor(State) = \{State, Country\}$ and $Descendant(State) = \{State, City\}$ in the schema tree shown in Figure 2.19 (a).

By using the flat approach, introduced in Section 2.2.1, it is possible to represent as multivalued dependencies the structural constraints embedded into a nested schema. For instance, if I is a nested relation of nested schema $Theater(Title)^*(Snack)^*$, then I satisfies the MVD $Theater \twoheadrightarrow Title$, since $TU(I)$ satisfies this MVD. Thus, $Theater \twoheadrightarrow Title$ is embedded into this nested schema. In general, given a nested schema S, the set of multivalued dependencies embedded into S, denoted by $MVD(S)$, is defined as:

$$MVD(S) \quad = \quad \{Ancestor(X) \twoheadrightarrow Descendant(Y) \mid (X, Y) \text{ is an edge in } SchemaTree(S)\}.$$

Thus, for example, the set of multivalued dependencies embedded into $Country(State(City)^*)^*$ is $\{Country \twoheadrightarrow \{State, City\}, \{Country, State\} \twoheadrightarrow City\}$.

Minimal covers for functional dependencies were introduced in Section 2.1.3 to synthesize 3NF database schemas from relation schemas. Minimal covers for multivalued dependencies are fundamental for the nested normal forms introduced by Özsoyoglu and Yuan [ÖY87, ÖY89]. We define them next. Given a set of multivalued dependencies Σ, an MVD $X \twoheadrightarrow Y \in \Sigma^+$ is said to be reduced if all the following conditions hold [ÖY87].

1. $X \twoheadrightarrow Y$ is not trivial.
2. $X \twoheadrightarrow Y$ is left-reduced, that is, there is no $X' \subsetneqq X$ such that $X' \twoheadrightarrow Y \in \Sigma^+$.
3. $X \twoheadrightarrow Y$ is right-reduced, that is, there is no $Y' \subsetneqq Y$ such that $X \twoheadrightarrow Y' \in \Sigma^+$ and $X \twoheadrightarrow Y'$ is not trivial.
4. There is no $X' \subsetneqq X$ such that $X' \twoheadrightarrow Y(X - X') \in \Sigma^+$.

Furthermore, Σ is said to be minimal if Σ is not equivalent to any of its proper subsets. Then, a set of multivalued dependencies Γ is a cover of Σ if $\Gamma^+ = \Sigma^+$. If in addition to this, Γ is minimal and every dependency in this set is reduced, then Γ is a *minimal cover* of Σ. Minimal covers for multivalued dependencies can be computed in polynomial time [ÖY87].

Finally, we are ready to present the first normal form introduced by Özsoyoglu and Yuan: NNF-87 [ÖY87]. This normal form defines four conditions that a well designed nested schema should satisfy. We examine these conditions in the context of the following example, introduced in Section 2.1.2. Assume that $U = \{\mathit{Title}, \mathit{Director}, \mathit{Theater}, \mathit{Snack}\}$ and $\Sigma = \{\mathit{Title} \twoheadrightarrow \mathit{Director}, \mathit{Theater} \twoheadrightarrow \mathit{Snack}\}$. Figure 2.20 shows three alternative ways of grouping U into a nested schema. Is any of these groupings a good representation of Σ?

None of the nested schemas shown in Figure 2.20 is a good representation of Σ. The schema tree of nested schema $\mathit{Title}(\mathit{Director})^*(\mathit{Theater})^*(\mathit{Snack})^*$ is shown in Figure 2.20 (a). The problem with this schema is that it has embedded some multivalued dependencies, such as $\mathit{Title} \twoheadrightarrow \mathit{Theater}$, that are not implied by Σ. The first condition in NNF-87 is that a nested schema S cannot contain more semantic information than Σ, that is, $\Sigma \models \mathit{MVD}(S)$. The schema tree of nested schema $S_1 = \mathit{Title}(\mathit{Director})^*(\mathit{Theater}(\mathit{Snack})^*)^*$ is shown in Figure 2.20 (b). Although Σ implies $\mathit{MVD}(S_1)$, S_1 is not a good representation of Σ since the multivalued dependency $\{\mathit{Title}, \mathit{Theater}\} \twoheadrightarrow \mathit{Snack} \in \mathit{MVD}(S_1)$ is not left-reduced in Σ (it can be deduced from $\mathit{Theater} \twoheadrightarrow \mathit{Snack}$). Indeed, this nested schema is prone to update anomalies. For example, let I be the following nested relation of S_1.

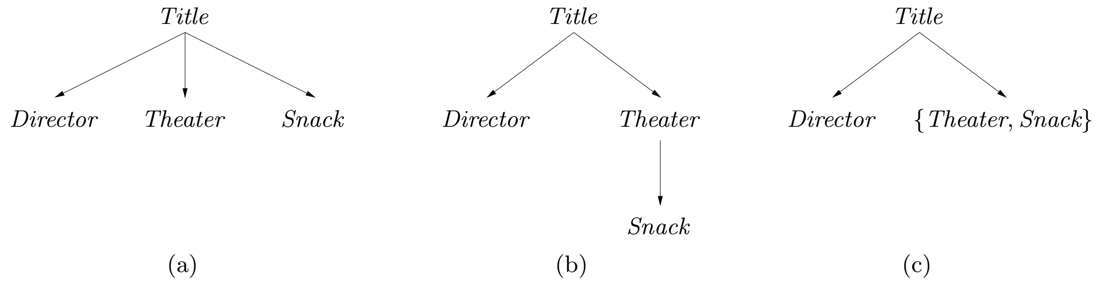


Figure 2.20: Three alternative representations of $\{\mathit{Title} \twoheadrightarrow \mathit{Director}, \mathit{Theater} \twoheadrightarrow \mathit{Snack}\}$.

<table>
<tr><td colspan="3">Title</td></tr>
<tr><td>Bad Company</td><td><table><tr><td>Director</td></tr><tr><td>Joel Schumacher</td></tr></table></td><td><table><tr><td colspan="2">Theater</td></tr><tr><td>Paramount</td><td><table><tr><td>Snack</td></tr><tr><td>coke</td></tr></table></td></tr></table></td></tr>
</table>

If we insert into this relation tuple (Insomnia, {Christopher Nolan}, {Paramount, {popcorn}}), then we will also have to modify both the original tuple in I and this new tuple in order to satisfy the MVD $\mathit{Theater} \twoheadrightarrow \mathit{Snack}$:

<table>
<tr><td colspan="3">Title</td></tr>
<tr><td>Bad Company</td><td><table><tr><td>Director</td></tr><tr><td>Joel Schumacher</td></tr></table></td><td><table><tr><td colspan="2">Theater</td></tr><tr><td>Paramount</td><td><table><tr><td>Snack</td></tr><tr><td>coke
popcorn</td></tr></table></td></tr></table></td></tr>
<tr><td>Insomnia</td><td><table><tr><td>Director</td></tr><tr><td>Christopher Nolan</td></tr></table></td><td><table><tr><td colspan="2">Theater</td></tr><tr><td>Paramount</td><td><table><tr><td>Snack</td></tr><tr><td>coke
popcorn</td></tr></table></td></tr></table></td></tr>
</table>

The second condition in NNF-87 forbids a nested schema S containing left-reducible or right-reducible dependencies in $MVD(S)$. Figure 2.20 (c) shows a third grouping of the set of attributes U corresponding to the nested schema $S_2 = \mathit{Title}(\mathit{Director})^*(\{\mathit{Theater}, \mathit{Snack}\})^*$. Although $\Sigma \models MVD(S_2)$ and every MVD embedded in S_2 is left-reduced and right-reduced, S_2 is not a good representation of Σ since the node $\{\mathit{Theater}, \mathit{Snack}\}$ has not been correctly split in order to represent the MVD

Theater $\rightarrow\!\!\!\rightarrow$ *Snack*. Indeed, this schema presents the same type of update anomalies that we show in the previous example. The attributes in the left hand side of Σ are a good indication of how to group U and they should be taken into account when designing a nested schema. This is the third condition in NNF-87 and it can be formalized as follows. Let S be a nested relation schema and Σ a set of multivalued dependencies over S. Assume that $\mathit{Attribute}(S) = U$. Then, the set of *keys* of Σ is defined as [ÖY87]:

$$\{X \mid \text{ there exist } Y \subseteq U \text{ such that } X \rightarrow\!\!\!\rightarrow Y \text{ is a reduced MVD in } \Sigma\}.$$

Notice that if Γ is a cover of Σ, then X is a key of Σ if and only if X is a key of Γ. For every $V \subseteq U$, the set of *fundamental keys* on V, denoted by $\mathit{FKey}(V)$, is defined as [ÖY87]:

$$\{V \cap X \mid X \in \mathit{LHS}(\Sigma),\ V \cap X \neq \emptyset \text{ and } \text{there is no } Y \in \mathit{LHS}(\Sigma) \text{ such that } \emptyset \neq V \cap Y \subsetneq V \cap X\}.$$

Recall that $\mathit{LHS}(\Sigma)$ stands for the set of left hand sides in Σ. If S is in NNF-87, then the root of $\mathit{SchemaTree}(S)$ is a key of Σ and for each other node X in this tree, if $\mathit{FKey}(\mathit{Descendant}(X)) \neq \emptyset$, then $X \in \mathit{FKey}(\mathit{Descendant}(X))$. This condition does not hold in the example shown above since $\mathit{Descendant}(\{\mathit{Theater}, \mathit{Snack}\}) = \{\mathit{Theater}, \mathit{Snack}\}$, $\mathit{FKey}(\{\mathit{Theater}, \mathit{Snack}\}) = \{\mathit{Theater}\}$ and $\{\mathit{Theater}, \mathit{Snack}\} \notin \mathit{FKey}(\{\mathit{Theater}, \mathit{Snack}\})$.

To completely define NNF-87, we need to introduce some additional terminology for the fourth condition in this normal form. Let S be a nested relation schema and Σ a set of multivalued dependencies over S. Let (Y, Z) be an edge in $\mathit{SchemaTree}(S)$ and X a key of Σ. Then, Z is *transitive redundant* with respect to X in S if $X \rightarrow\!\!\!\rightarrow \mathit{Descendant}(Z) \in \Sigma^+$ and there exists sibling nodes $Z_1, \ldots, Z_n$ of Z in $\mathit{SchemaTree}(S)$ such that the following conditions hold:

$$X \subseteq \mathit{Ancestor}(Y) \cup \bigcup_{i=1}^{n} \mathit{Descendant}(Z_i),$$

$$X \cup \bigcup_{i=1}^{n} \mathit{Descendant}(Z_i) \rightarrow\!\!\!\rightarrow \mathit{Ancestor}(Y) \notin \Sigma^+.$$

Then, (S, Σ) is in NNF-87 [ÖY87] if there exists a minimal cover Γ of Σ such that the following conditions hold.

1. $\Gamma \models \mathit{MVD}(S)$.

2. Every multivalued dependency in $MVD(S)$ is left- and right-reduced in Γ.

3. The root of $SchemaTree(S)$ is a key of Γ and for each other node X in $SchemaTree(S)$, if $FKey(Descendant(X)) \neq \emptyset$, then $X \in FKey(Descendant(X))$.

4. For each node X in $SchemaTree(S)$, there is no key Y of Γ such that X is transitive redundant with respect to Y in S.

A nested database schema is in NNF-87 if every nested relation schema in it is in NNF-87.

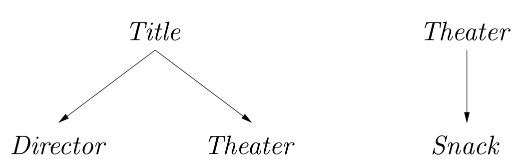
So far, we have not mentioned how to deal with functional dependencies in the definition of NNF-87. If Σ contains functional dependencies and multivalued dependencies, then every functional dependency $X \rightarrow Y$ is replaced by $\{X \rightarrow\!\!\!\rightarrow A \mid A \in Y\}$ in order to test whether (S, Σ) is in NNF-87. Özsoyoglu and Yuan [ÖY89] introduced a second normal form, namely NNF-89, that is defined in terms of the same conditions than NNF-87, except that if Σ contains functional dependencies and multivalued dependencies, then the envelope of Σ (see Section 2.1.3) is used to test whether (S, Σ) is in NNF-89.

Özsoyoglu and Yuan [ÖY87] present a NNF-87 decomposition algorithm for nested relational databases. We do not present this algorithm here, we just point out some of its important characteristics. The input of this algorithm is a set of attributes U and a minimal set of multivalued dependencies Σ containing only reduced MVDs. The output of this algorithm is a nested database schema $S' = \{S_i \mid i \in [1, n]\}$ such that:

- S' is a lossless decomposition of S, that is, $U = Attribute(S_1) \cup \cdots \cup Attribute(S_n)$ and $\Sigma \models \bowtie[Attribute(S_1), \ldots, Attribute(S_n)]$.

- S' is in NNF-87: For every $i \in [1, n]$, (S_i, Σ_i) is in NNF-87, where Σ_i is the projection of Σ over $Attribute(S_i)$, that is, $\{X \rightarrow\!\!\!\rightarrow Y \cap Attribute(S_i) \mid X \rightarrow\!\!\!\rightarrow Y \in \Sigma^+ \text{ and } X \subseteq Attribute(S_i)\}$.

For example, if $U = \{Title, Director, Theater, Snack\}$ and $\Sigma = \{Title \rightarrow\!\!\!\rightarrow Director, Theater \rightarrow\!\!\!\rightarrow Snack\}$, then the output of the algorithm is the following nested database schema.



Observe that $Title \rightarrow\!\!\!\rightarrow Theater$ is embedded in the first nested schema. This is not a mistake, it represents the following fact. Given that $Title \rightarrow\!\!\!\rightarrow \{Theater, Snack\}$ is implied by Σ, if a database instance I defined over U satisfies Σ, then $\pi_{\{\mathit{Title},\, \mathit{Director},\, \mathit{Theater}\}}(I)$ $\models$ $\mathit{Title} \rightarrow\!\!\!\rightarrow \mathit{Theater}$. Thus, to understand when the decomposed nested schema $S' = \{S_i \mid i \in [1, n]\}$ is a dependency preserving decomposition of Σ, we have to define how to transform the MVDs in $\mathit{MVD}(S_i)$ ($i \in [1, n]$) into MVDs over the set of attributes U. Formally, for every $i \in [1, n]$, $\mathit{MVD}_U(S_i)$ is defined as follows.

$$\{X \rightarrow\!\!\!\rightarrow Y \in \Sigma \mid \text{there exists } X \rightarrow\!\!\!\rightarrow Z \in \mathit{MVD}(S_i) \text{ such that } Z = Y \cap \mathit{Attribute}(S_i)\}.$$

Then, S' is a dependency preserving decomposition of Σ if $\mathit{MVD}_U(S_1) \cup \cdots \cup \mathit{MVD}_U(S_n) \models \Sigma$. It was shown in [ÖY87] that if Σ is conflict-free (see Section 2.1.2), then S' is a dependency preserving decomposition of S.

NNF-96

Mok et al. [MNE96] introduced a nested normal form for precisely characterizing redundancy in nested relational databases. This normal form is defined as follows. Let S be a nested relation schema and Σ a set of functional dependencies and multivalued dependencies over S. Then, (S, Σ) is in NNF-96 [MNE96] if the following conditions hold.

1. Σ is equivalent to $\mathit{MVD}(S) \cup \{X \rightarrow Y \mid X \rightarrow Y \in \Sigma^+\}$.
2. For every nontrivial FD $X \rightarrow A \in \Sigma^+$, $X \rightarrow \mathit{Ancestor}(N_A)$ is also in Σ^+, where N_A is the node in $\mathit{SchemaTree}(S)$ that contains attribute A.

Mok et al. [MNE96] proved that nested relation schemas in NNF-96 cannot contain redundant information. To present this result, we need to introduce some terminology. Let S be a nested relation schema and Σ a set of functional and multivalued dependencies over S. Then, (S, Σ) is *consistent* if $\Sigma \models \mathit{MVD}(S)$, where $\mathit{MVD}(S)$ is the set of multivalued dependencies embedded into S (see previous section for a formal definition). Furthermore, given an instance I of S, a tuple t in I and an atomic attribute A, $t[A]$ is *redundant in* I [MNE96] if for every $a \neq t[A]$, the instance I' obtained by replacing $t[A]$ by a does not satisfy Σ. Mok et al. [MNE96] showed that if (S, Σ) is a consistent nested relation schema, then (S, Σ) is not redundant if and only if (S, Σ) is in NNF-96. Interestingly, Vincent's characterization of 4NF [Vin99] (see Section 2.1.3) is a corollary of this result, since every relation schema $S = (R[U], \Sigma)$ is consistent ($\mathit{MVD}(S) = \{U \rightarrow\!\!\!\rightarrow U\}$ is a trivial set of MVDs) and S is in 4NF if and only if S is in NNF-96 [MNE96].

Mok [Mok02] introduced an NNF-96 normalization algorithm that works under the assumption that the set of multivalued dependencies contained in Σ is conflict-free. The input of this algorithm is a set of attributes U, the set of functional dependencies contained in Σ and a join dependency equivalent to the set of multivalued dependencies contained in Σ [12]. The output of this algorithm is a lossless and dependency preserving decomposition of (S, Σ) in NNF-96.

What is the relationship between NNF-87 and NNF-96? Mok [Mok02] showed that if Σ is a conflict-free set of multivalued dependencies and (S, Σ) is in NNF-87, then (S, Σ) is in NNF-96. Mok also showed that the converse of this theorem is not true.

[12] Such a join dependency exists since the set of multivalued dependencies in Σ is conflict-free [BFMY83].

Chapter 3

An Information-Theoretic Approach to Normal Forms

Normalization as a way of producing good relational database designs is a well-understood topic. However, the same problem of distinguishing well-designed databases from poorly designed ones arises in other data models, in particular, XML. While in the relational world the criteria for being well-designed are usually very intuitive and clear to state, they become more obscure when one moves to more complex data models.

Our goal in this chapter is to provide a set of tools for testing when a condition on a database design, specified by a *normal form*, corresponds to a good design. We use techniques of information theory, and define a measure of information content of elements in a relational database with respect to a set of constraints. We use this measure to provide information-theoretic justification for familiar relational normal forms such as BCNF, 4NF, PJ/NF, 5NFR, DK/NF. We then look at information-theoretic criteria for justifying normalization algorithms for relational databases.

The information-theoretic measure introduced in this chapter is robust; even though it is defined in the context of relational databases, it can be extended straightforwardly to different data model such as nested relational and XML. In particular, in Chapter 7, we introduce a normal form for XML documents and we use this measure to justify it. In that chapter, we also look at information-theoretic criteria for justifying normalization algorithms for XML databases.

3.1 Introduction

What constitutes a good database design? This question has been studied extensively, with well-known solutions presented in practically all database texts. But what is it that makes a database design good? This question is usually addressed at a much less formal level. For instance, we know that BCNF is an example of a good design, and we usually say that this is because BCNF eliminates update anomalies. Most of the time this is sufficient, given the simplicity of the relational model and our good intuition about it.

Several papers (see Section 2.1.4) attempted a more formal evaluation of normal forms, by relating it to the elimination of update anomalies. Another criterion is the existence of algorithms that produce good designs: for example, we know that every database scheme can be losslessly decomposed into one in BCNF, but some constraints may be lost along the way.

The previous work was specific for the relational model. As new data formats such as XML are becoming critically important, classical database theory problems have to be revisited in the new context [Wid99, Via01, Suc01, BFSW01]. However, there is as yet no consensus on how to address the problem of well-designed data in the XML setting [EM01a, AL02].

It is problematic to evaluate XML normal forms based on update anomalies; while some proposals for update languages exist [TIHW01], no XML update language has been standardized. Likewise, using the existence of good decomposition algorithms as a criterion is problematic: for example, to formulate losslessness, one needs to fix a small set of operations in some language, that would play the same role for XML as relational algebra for relations.

This suggests that one needs a different approach to the justification of normal forms and good designs. Such an approach must be applicable to new data models *before* the issues of query/update/constraint languages for them are completely understood and resolved. Therefore, such an approach must be based on some intrinsic characteristics of the data, as opposed to query/update languages for a particular data model. In this chapter we suggest such an approach based on information-theoretic concepts, more specifically, on measuring the information content of the data. Our goal here is to introduce information-theoretic measures of "goodness" of a design, and test them in the relational world. To be applicable in other contexts, we expect these measures to characterize familiar normal forms. We also use our measures to reason about normalization algorithms for relational databases, by showing that standard decomposition algorithms never decrease the information content of any piece of data in a database/document.

The rest of this chapter is organized as follows. In Section 3.2 we give the notations, and review the basics of information theory (entropy and conditional entropy). Section 3.3 is an "appetizer" for the main part of the chapter: we present a particularly simple information-theoretic way of measuring the information content of a database, and show how it characterizes BCNF and 4NF. The measure, however, is too coarse, and, furthermore, cannot be used to reason about normalization algorithms. In Section 3.4 we present our main information-theoretic measure of the information content of a database. Unlike the measure studied before [Lee87, CP87, DR00, LL03], our measure takes into account both database instance and schema constraints, and defines the content with respect to a set of constraints. A well-designed database is one in which the content of each datum is the maximum possible. We use this measure to characterize BCNF and 4NF as the best way to design schemas under FDs and MVDs, and to justify normal forms involving JDs (PJ/NF, 5NFR) and other types of integrity constraints (DK/NF). Finally, in Section 3.5, we use the measures of Section 3.4 to reason about normalization algorithms for relational databases, by showing that good normalization algorithms do not decrease the information content of each datum at every step.

3.2 Notations

In this chapter we assume familiarity with the terminology introduced in Section 2.1.

3.2.1 Schemas and Instances

Given a relation schema R, we denote by $sort(R)$ the set of attributes of R. We shall identify $sort(R)$ of cardinality m with $\{1, \ldots, m\}$. Throughout this chapter, we assume that the domain of each attribute is $\mathbb{N}^+$, the set of positive integers. Thus, an instance I of schema S assigns to each symbol $R \in S$ with $m = |sort(R)|$ a relation $I(R)$ which is a finite set of m-tuples over $\mathbb{N}^+$. By $adom(I)$ we mean the active domain of I, that is, the set of all elements of $\mathbb{N}^+$ that occur in I. The size of $I(R)$ is defined as $\|I(R)\| = |sort(R)| \cdot |I(R)|$, and the size of I is $\|I\| = \sum_{R \in S} \|I(R)\|$. If I is an instance of S, the set of *positions* in I, denoted by $Pos(I)$, is the set $\{(R, t, A) \mid R \in S,\ t \in I(R)$ and $A \in sort(R)\}$. Note that $|Pos(I)| = \|I\|$. Furthermore, given a relational schema S

and a set of data dependencies Σ over S, we define $inst(S, \Sigma)$ as the set of all database instances of S satisfying Σ and $inst_k(S, \Sigma)$ as $\{I \in inst(S, \Sigma) \mid adom(I) \subseteq [1, k]\}$, where $[1, k] = \{1, \ldots, k\}$.

3.2.2 Basics of Information Theory

The main concept of information theory is that of entropy, which measures the amount of information provided by a certain event. Assume that an event can have n different outcomes $s_1, \ldots, s_n$, each with probability p_i, $i \leq n$. How much information is gained by knowing that s_i occurred? This is clearly a function of p_i. Suppose g measures this information; then it must be continuous and decreasing function with domain $(0, 1]$ (the higher the probability, the less information gained) and $g(1) = 0$ (no information is gained if the outcome is known in advance). Furthermore, g is additive: if outcomes are independent, the amount of information gained by knowing two successive outcomes must be the sum of the two individuals amounts, that is, $g(p_i \cdot p_j) = g(p_i) + g(p_j)$. The only function satisfying these conditions is $g(x) = -c \ln x$, where c is an arbitrary positive constant [Sha48]. It is customary to use base 2 logarithms: $g(x) = -\log x$.

The *entropy* of a probability distribution represents the average amount of information gained by knowing that a particular event occurred. Let $\mathcal{A} = (\{s_1, \ldots, s_n\}, P_{\mathcal{A}})$ be a probability space. If $p_i = P_{\mathcal{A}}(s_i)$, then the entropy of $\mathcal{A}$, denoted by $H(\mathcal{A})$, is defined to be

$$H(\mathcal{A}) \;=\; \sum_{i=1}^{n} p_i \log \frac{1}{p_i} \;=\; -\sum_{i=1}^{n} p_i \log p_i.$$

Observe that some of the probabilities in the space $\mathcal{A}$ can be zero. For that case, we adopt the convention that $0 \log \frac{1}{0} = 0$, since $\lim_{x \to 0} x \log \frac{1}{x} = 0$. It is known that $0 \leq H(\mathcal{A}) \leq \log n$, with $H(\mathcal{A}) = \log n$ only for the uniform distribution $P_{\mathcal{A}}(s_i) = 1/n$ [CT91].

We shall also use *conditional entropy*. Assume that we are given two probability spaces $\mathcal{A} = (\{s_1, \ldots, s_n\}, P_{\mathcal{A}})$, $\mathcal{B} = (\{s'_1, \ldots, s'_m\}, P_{\mathcal{B}})$ and, furthermore, we know probabilities $P(s'_j, s_i)$ of all the events (s'_j, s_i) (that is, $P_{\mathcal{A}}$ and $P_{\mathcal{B}}$ need not be independent). Then the conditional entropy of $\mathcal{B}$ given $\mathcal{A}$, denoted by $H(\mathcal{B} \mid \mathcal{A})$, gives the average amount of information provided by $\mathcal{B}$ if $\mathcal{A}$ is known [CT91]. It is defined using conditional probabilities $P(s'_j \mid s_i) = P(s'_j, s_i)/P_{\mathcal{A}}(s_i)$:

$$H(\mathcal{B} \mid \mathcal{A}) = \sum_{i=1}^{n} \left(P_{\mathcal{A}}(s_i) \sum_{j=1}^{m} P(s'_j \mid s_i) \log \frac{1}{P(s'_j \mid s_i)} \right).$$

A	B	C
1	2	3
1	2	4

(a)

A	B	C
1	1	2
2	3	4

(b)

A	B	C
1	2	3
1	2	4
1	2	5

(c)

Figure 3.1: Database instances.

3.3 Information Theory and Normal Forms: an Appetizer

We will now see a particularly simple way to provide information-theoretic characterization of normal forms. Although it is very easy to present, it has a number of shortcomings, and a more elaborate measure will be presented in the next section.

Violating a normal form, e.g., BCNF, implies having redundancies. For example, if $S = \{R(A, B, C)\}$ and $\Sigma = \{A \to B\}$, then (S, Σ) is not in BCNF (A is not a key) and some instances can contain redundant information: in Figure 3.1 (a), the value of the gray cell must be equal to the value below it. We do not need to store this value as it can be inferred from the remaining values and the constraints.

We now use the concept of entropy to measure the information content of every position in an instance of S. The basic idea is as follows: we measure how much information we gain if we lose the value in a given position, and then someone restores it (either to the original, or to some other value, not necessarily from the active domain). For instance, if we lose the value in the gray cell in Figure 3.1 (a), we gain zero information if it gets restored, since we know from the rest of the instance and the constraints that it equals 2. Formally, let $I \in inst_k(S, \Sigma)$ (that is, $adom(I) \subseteq [1, k]$) and let $p \in Pos(I)$ be a position in I. For any value a, let $I_{p \leftarrow a}$ be a database instance constructed from I by replacing the value in position p by a. We define a probability space $\mathcal{E}^k_\Sigma(I, p) = ([1, k+1], P)$ and use its entropy as the measure of information in p (we define it on $[1, k+1]$ to guarantee that there is at least one value outside of the active domain). The function P is given by:

$$P(a) = \begin{cases} 0 & I_{p \leftarrow a} \not\models \Sigma, \\ 1/|\{b \mid I_{p \leftarrow b} \models \Sigma\}| & \text{otherwise.} \end{cases}$$

In other words, let m be the number of $b \in [1, k+1]$ such that $I_{p \leftarrow b} \models \Sigma$ (note that $m > 0$ since $I \models \Sigma$). For each such b, $P(b) = 1/m$, and elsewhere $P = 0$. For example, for the instance in Figure 3.1 (a) if p is the position of the gray cell, then the probability distribution is as follows: $P(2) = 1$ and $P(a) = 0$, for all other $a \in [1, k+1]$. Thus, the entropy of $\mathcal{E}^k_\Sigma(I, p)$ for position p is zero, as we expect. More generally, we can show the following.

Theorem 3.3.1 *Let Σ be a set of FDs (or FDs and MVDs) over a schema S. Then (S, Σ) is in BCNF (or 4NF, resp.) if and only if for every $k > 1$, $I \in inst_k(S, \Sigma)$ and $p \in Pos(I)$,*

$$H(\mathcal{E}^k_\Sigma(I, p)) > 0.$$

Proof: We give the proof for the case of FDs; for FDs and MVDs the proof is almost identical.

($\Rightarrow$) Assume that (S, Σ) is in BCNF. Fix $k > 0$, $I \in inst_k(S, \Sigma)$ and $p \in Pos(I)$. Assume that a is the p-th element in I. We show that $I_{p \leftarrow k+1} \models \Sigma$, from which we conclude that $H(\mathcal{E}^k_\Sigma(I, p)) > 0$, since $\mathcal{E}^k_\Sigma(I, p)$ is uniformly distributed, and $P(a), P(k+1) \neq 0$.

Towards a contradiction, assume that $I_{p \leftarrow k+1} \not\models \Sigma$. Then there exist $R \in S$, $t'_1, t'_2 \in I_{p \leftarrow k+1}(R)$ and $X \rightarrow A \in \Sigma^+$ such that $t'_1[X] = t'_2[X]$ and $t'_1[A] \neq t'_2[A]$. Assume that t'_1, t'_2 were generated from tuples $t_1, t_2 \in I(R)$ (hence $t_1 \neq t_2$), respectively. Note that $t'_1[X] = t_1[X]$ (if $t_1[X] \neq t'_1[X]$, then $t'_1[B] = k+1$ for some $B \in X$; given that $k+1 \notin adom(I)$, only one position in $I_{p \leftarrow k+1}$ mentions this value and, therefore, $t'_1[X] \neq t'_2[X]$, a contradiction). Similarly, $t'_2[X] = t_2[X]$ and, therefore, $t_1[X] = t_2[X]$. Given that (S, Σ) is in BCNF, X must be a key in R. Hence, $t_1 = t_2$, since $I \models \Sigma$, which is a contradiction.

($\Leftarrow$) Assume that (S, Σ) is not in BCNF. We show that there exists $k > 0$, $I \in inst_k(S, \Sigma)$ and $p \in Pos(I)$ such that $H(\mathcal{E}^k_\Sigma(I, p)) = 0$. Since (S, Σ) is not in BCNF, there exist $R \in S$ and $X \rightarrow A \in \Sigma^+$ such that $A \notin X$, $X \cup \{A\} \subsetneq sort(R)$ and X is not a key in R. Thus, there exists a database instance I of S such that $I \models \Sigma$ and $I \not\models X \rightarrow sort(R)$. We can assume that $I(R)$ contains only two tuples, say t_1, t_2. Let k be the greatest value in I, $i = t_1[A]$ and p be the position of $t_1[A]$ in I. It is easy to see that $I \in inst_k(S, \Sigma)$ and $P(j) = 0$, for every $j \neq i$ in $[1, k+1]$, since $t_1[A]$ must be equal to $t_2[A] = i$. Therefore, $H(\mathcal{E}^k_\Sigma(I, p)) = 0$. $\square$

We note that this theorem is essentially equivalent to Vincent's characterizations of BCNF and 4NF [Vin99] (see Section 2.1.3).

Theorem 3.3.1 says that a schema is in BCNF or 4NF iff for every instance, each position carries non-zero amount of information. This is a clean characterization of BCNF and 4NF, but the measure $H(\mathcal{E}^k_\Sigma(I, p))$ is not accurate enough for a number of reasons. For example, let $\Sigma_1 = \{A \rightarrow B\}$ and $\Sigma_2 = \{A \rightarrow\rightarrow B\}$. The instance I in Figure 3.1(a) satisfies Σ_1 and Σ_2. Let p be the position of the gray cell in I. Then $H(\mathcal{E}^k_{\Sigma_1}(I, p)) = H(\mathcal{E}^k_{\Sigma_2}(I, p)) = 0$. But intuitively, the information content of p must be higher under Σ_2 than Σ_1, since Σ_1 says that the value in p must be equal to the value below it, and Σ_2 says that this should only happen if the values of the C-attribute are distinct.

Next, consider I_1 and I_2 shown in Figures 3.1 (a) and (c), respectively. Let $\Sigma = \{A \rightarrow B\}$, and let p_1 and p_2 denote the positions of the gray cells in I_1 and I_2. Then $H(\mathcal{E}^k_\Sigma(I_1, p_1)) = H(\mathcal{E}^k_\Sigma(I_2, p_2)) = 0$. But again we would like them to have different values, as the amount of redundancy is higher in I_2 than in I_1. Finally, let $S = R(A, B)$, $\Sigma = \{\emptyset \rightarrow\rightarrow A\}$, and $I = \{1, 2\} \times \{3, 4\} \in inst(S, \Sigma)$. For each position, the entropy would be zero. However, consider both positions in attribute A corresponding to the value 1. If they both disappear, then we know that no matter how they are restored, the values must be the same. The measure presented in this section cannot possibly talk about inter-dependencies of this kind.

In the next section we will present a measure that overcomes these problems.

3.4 A General Definition of Well-Designed Data

Let S be a schema, Σ a set of constraints, and $I \in inst(S, \Sigma)$ an instance with $\|I\| = n$. Recall that $Pos(I)$ is the set of positions in I, that is, $\{(R, t, A) \mid R \in S,\ t \in I(R)$ and $A \in sort(R)\}$. Our goal is to define a function $\text{INF}_I(p \mid \Sigma)$, the information content of a position $p \in Pos(I)$ with respect to the set of constraints Σ. For a general definition of well-designed data, we want to say that this measure has the maximum possible value. This is a bit problematic for the case of an infinite domain ($\mathbb{N}^+$), since we only know what the maximum value of entropy is for a discrete distribution over k elements: $\log k$.

A	B	C
6	5	4
3	2	1

(a) An enumeration of I

A	B	C
1	7	3
1	2	4

(b) $I_{(7,\bar{a}_1)} = \sigma_1(I_{(7,\bar{a}_1)})$

A	B	C
v_6	7	3
1	2	v_1

(c) $I_{(7,\bar{a}_2)}$

A	B	C
8	7	3
1	2	4

(d) $\sigma_2(I_{(7,\bar{a}_2)})$

Figure 3.2: Defining $\text{INF}_I^k(p \mid \Sigma)$.

To overcome this, we define, for each $k > 0$, a function $\text{INF}_I^k(p \mid \Sigma)$ that would only apply to instances whose active domain is contained in $[1, k]$, and then consider the ratio $\text{INF}_I^k(p \mid \Sigma)/\log k$. This ratio tells us how close the given position p is to having the maximum possible information content, for databases with active domain in $[1, k]$. As our final measure $\text{INF}_I(p \mid \Sigma)$ we then take the limit of this sequence as k goes to infinity.

Informally, $\text{INF}_I^k(p \mid \Sigma)$ is defined as follows. Let $X \subseteq Pos(I) - \{p\}$. Suppose the values in those positions X are lost, and then someone restores them from the set $[1, k]$; we measure how much information about the value in p this gives us. This measure is defined as the entropy of a suitably chosen distribution. Then $\text{INF}_I^k(p \mid \Sigma)$ is the average such entropy over all sets $X \subseteq Pos(I) - \{p\}$. Note that this is much more involved than the definition of the previous section, as it takes into account all possible interactions between different positions in an instance and the constraints.

We now present this measure formally. An *enumeration* of I with $\|I\| = n$, $n > 0$, is a bijection f_I between $Pos(I)$ and $[1, n]$. From now on, we assume that every instance has an associated enumeration[1]. We say that the position of $(R, t, A) \in Pos(I)$ is p in I if the enumeration of I assigns p to (R, t, A), and if R is clear from the context, we say that the position of $t[A]$ is p. We normally associate positions with their rank in the enumeration f_I.

Fix a position $p \in Pos(I)$. As the first step, we need to describe all possible ways of removing values in a set of positions X, different from p. To do this, we shall be placing variables from a set $\{v_i \mid i \geq 1\}$ in positions where values are to be removed, where v_i can occur only in position i. Furthermore, we assume that each set of positions is equally likely to be removed. To model this, let $\Omega(I, p)$ be the set of all 2^{n-1} vectors $(a_1, \ldots, a_{p-1}, a_{p+1}, \ldots, a_n)$ such that for every $i \in [1, n] - \{p\}$, a_i is either v_i or the value in the

[1] The choice of a particular enumeration will not affect the measures we define.

i-th position of I. A probability space $\mathcal{A}(I,p) = (\Omega(I,p), P)$ is defined by taking P to be the uniform distribution.

Example 3.4.1 Let I be the database instance shown in Figure 3.1 (a). An enumeration of the positions in I is shown in Figure 3.2 (a). Assume that p is the position of the gray cell shown in Figure 3.1 (a), that is, $p = 5$. Then $\bar{a}_1 = (4, 2, 1, 3, 1)$ and $\bar{a}_2 = (v_1, 2, 1, 3, v_6)$ are among the 32 vectors in $\Omega(I,p)$. For each of these vectors, we define P as $\frac{1}{32}$. $\square$

Our measure $\text{INF}_I^k(p \mid \Sigma)$, for $I \in inst_k(S, \Sigma)$, will be defined as the conditional entropy of a distribution on $[1, k]$, given the above distribution on $\Omega(I,p)$. For that, we define conditional probabilities $P(a \mid \bar{a})$ that characterize how likely a is to occur in position p, if some values are removed from I according to the tuple $\bar{a}$ from $\Omega(I,p)$ [2]. We need a couple of technical definitions first. If $\bar{a} = (a_i)_{i \neq p}$ is a vector in $\Omega(I,p)$ and $a > 0$, then $I_{(a,\bar{a})}$ is a table obtained from I by putting a in position p, and a_i in position $i, i \neq p$. If $k > 0$, then a *substitution* $\sigma : \bar{a} \to [1, k]$ assigns a value from $[1, k]$ to each a_i which is a variable, and leaves the other a_i values intact. We can extend σ to $I_{(a,\bar{a})}$ and thus talk about $\sigma(I_{(a,\bar{a})})$.

Example 3.4.2 (example *3.4.1* continued) Let $k = 8$ and σ_1 be an arbitrary substitution from $\bar{a}_1$ to $[1, 8]$. Note that σ_1 is the identity substitution, since $\bar{a}_1$ contains no variables. Figure 3.2 (b) shows $I_{(7,\bar{a}_1)}$, which is equal to $\sigma_1(I_{(7,\bar{a}_1)})$. Let σ_2 be a substitution from $\bar{a}_2$ to $[1, 8]$ defined as follows: $\sigma(v_1) = 4$ and $\sigma(v_6) = 8$. Figure 3.2 (c) shows $I_{(7,\bar{a}_2)}$ and Figure 3.2 (d) shows the database instance generated by applying σ_2 to $I_{(7,\bar{a}_2)}$. $\square$

If Σ is a set of constraints over S, then $SAT_\Sigma^k(I_{(a,\bar{a})})$ is defined as the set of all substitutions $\sigma : \bar{a} \to [1, k]$ such that $\sigma(I_{(a,\bar{a})}) \models \Sigma$ and $\|\sigma(I_{(a,\bar{a})})\| = \|I\|$ (the latter ensures that no two tuples collapse as the result of applying σ). With this, we define $P(a \mid \bar{a})$ as:

$$P(a \mid \bar{a}) \;\; = \;\; \frac{|SAT_\Sigma^k(I_{(a,\bar{a})})|}{\sum\limits_{b \in [1,k]} |SAT_\Sigma^k(I_{(b,\bar{a})})|}.$$

[2] We use the same letter P here, but this will never lead to confusion. Furthermore, all probability distributions depend on I, p, k and Σ, but we omit them as parameters of P since they will always be clear from the context.

We remark that this corresponds to conditional probabilities with respect to a distribution P' on $[1,k] \times \Omega(I,p)$ defined by $P'(a,\bar{a}) = P(a \mid \bar{a}) \cdot (1/2^{n-1})$, and that P' is indeed a probability distribution for every $I \in inst_k(S,\Sigma)$ and $p \in Pos(I)$.

Example 3.4.3 (example *3.4.2* continued) Assume that $\Sigma = \{A \rightarrow B\}$. Given that the only substitution σ from $\bar{a}_1$ to $[1,8]$ is the identity, for every $a \in [1,8]$, $a \neq 2$, $\sigma(I_{(a,\bar{a}_1)}) \not\models \Sigma$, and, therefore, $SAT^8_\Sigma(I_{(a,\bar{a}_1)}) = \emptyset$. Thus, $P(2 \mid \bar{a}_1) = 1$ since $\sigma(I_{(2,\bar{a}_1)}) \models \Sigma$. This value reflects the intuition that if the value in the gray cell of the instance shown in Figure 3.1 (a) is removed, then it can be inferred from the remaining values and the FD $A \rightarrow B$.

There are 64 substitutions with domain $\bar{a}_2$ and range $[1,8]$. A substitution σ is in $SAT^8_\Sigma(I_{(7,\bar{a}_2)})$ if and only if $\sigma(v_6) \neq 1$, and, therefore, $|SAT^8_\Sigma(I_{(7,\bar{a}_2)})| = 56$. The same can be proved for every $a \in [1,8]$, $a \neq 2$. On the other hand, the only substitution that is not in $SAT^8_\Sigma(I_{(2,\bar{a}_2)})$ is $\sigma(v_1) = 3$ and $\sigma(v_6) = 1$, since $\sigma(I_{(2,\bar{a}_2)})$ contains only one tuple. Thus, $|SAT^8_\Sigma(I_{(2,\bar{a}_2)})| = 63$ and, therefore,

$$P(a \mid \bar{a}_2) = \begin{cases} \frac{63}{455} & \text{if } a = 2, \\ \\ \frac{56}{455} & \text{otherwise.} \end{cases}$$

□

We define a probability space $\mathcal{B}^k_\Sigma(I,p) = ([1,k], P)$ where

$$P(a) \quad = \quad \frac{1}{2^{n-1}} \sum_{\bar{a} \in \Omega(I,p)} P(a \mid \bar{a}) \ ,$$

and, again, omit I, p, k and Σ as parameters, and overload the letter P since this will never lead to confusion.

The measure of the amount of information in position p, $\text{INF}^k_I(p \mid \Sigma)$, is the conditional entropy of $\mathcal{B}^k_\Sigma(I,p)$ given $\mathcal{A}(I,p)$, that is, the average information provided by p, given all possible ways of removing values in the instance I:

$$\text{INF}^k_I(p \mid \Sigma) \ \stackrel{\text{def}}{=} \ H(\mathcal{B}^k_\Sigma(I,p) \mid \mathcal{A}(I,p)) \ = \ \sum_{\bar{a} \in \Omega(I,p)} \left(P(\bar{a}) \sum_{a \in [1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \right).$$

Note that for $\bar{a} \in \Omega(I,p)$, $\sum_{a \in [1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a|\bar{a})}$ measures the amount of information in position p, given a set of constraints Σ and some missing values in I, represented by the variables in $\bar{a}$. Thus, $\textsc{Inf}_I^k(p \mid \Sigma)$ is the average such amount over all $\bar{a} \in \Omega(I, p)$. Furthermore, from the definition of conditional entropy, $0 \leq \textsc{Inf}_I^k(p \mid \Sigma) \leq \log k$, and the measure $\textsc{Inf}_I^k(p \mid \Sigma)$ depends on the domain size k. We now consider the ratio of $\textsc{Inf}_I^k(p \mid \Sigma)$ and the maximum entropy $\log k$. It turns out that this sequence converges:

Lemma 3.4.4 *If Σ is a set of first-order constraints over a schema S, then for every $I \in inst(S, \Sigma)$ and $p \in Pos(I)$, $\lim_{k\to\infty} \textsc{Inf}_I^k(p \mid \Sigma)/\log k$ exists.*

The proof of this lemma is given in Appendix A.1. In fact, Lemma 3.4.4 shows that such a limit exists for any set of *generic* constraints, that is, constraints that do not depend on the domain. This finally gives us the definition of $\textsc{Inf}_I(p \mid \Sigma)$.

Definition 3.4.5 *For $I \in inst(S, \Sigma)$ and $p \in Pos(I)$, the measure $\textsc{Inf}_I(p \mid \Sigma)$ is defined as*

$$\lim_{k\to\infty} \frac{\textsc{Inf}_I^k(p \mid \Sigma)}{\log k} .$$

$\textsc{Inf}_I(p \mid \Sigma)$ measures how much information is contained in position p, and $0 \leq \textsc{Inf}_I(p \mid \Sigma) \leq 1$. A well-designed schema should not have an instance with a position that has less than maximum information:

Definition 3.4.6 *A database specification (S, Σ) is* well-designed *if for every $I \in inst(S, \Sigma)$ and every $p \in Pos(I)$, $\textsc{Inf}_I(p \mid \Sigma) = 1$.*

Example 3.4.7 Let S be a database schema $\{R(A,\ B,\ C)\}$. Let $\Sigma_1 = \{A \to BC\}$. Figure 3.1 (b) shows an instance I of S satisfying Σ_1 and Figure 3.3 (a) shows the value of $\textsc{Inf}_I^k(p \mid \Sigma_1)$ for $k = 5, 6, 7$, where p is the position of the gray cell. As expected, the value of $\textsc{Inf}_I^k(p \mid \Sigma_1)$ is maximal, since (S, Σ_1) is in BCNF. Indeed, given that we have to preserve the number of tuples, the A-values must be distinct, hence all possibilities for selecting B and C are open.

The next two examples show that the measure $\textsc{Inf}_I^k(p \mid \Sigma)$ can distinguish cases that were indistinguishable with the measure of Section 3.3. Let $\Sigma_2 = \{A \to B\}$ and $\Sigma_2' = \{A \to\!\!\to B\}$. Figure 3.1 (a) shows an instance I of S satisfying both Σ_2 and Σ_2'. Figure 3.3 (b) shows the value of $\textsc{Inf}_I^k(p \mid \Sigma_2)$ and $\textsc{Inf}_I^k(p \mid \Sigma_2')$ for $k = 5, 6, 7$. As expected, the values are smaller for Σ_2. Finally, let $\Sigma_3 = \{A \to B\}$. Figures 3.1 (a) and 3.1 (c) show instances I_1, I_2 of S satisfying Σ_3. We expect the information content of the gray cell to be smaller in I_2 than in I_1, but the measure used in Section 3.3 could not

k	$A \to BC$	$\log k$
5	2.3219	2.3219
6	2.5850	2.5850
7	2.8074	2.8074

(a)

k	$A \to B$	$A \to\to B$
5	2.0299	2.2180
6	2.2608	2.4637
7	2.4558	2.6708

(b)

k	I_1	I_2
5	2.0299	1.8092
6	2.2608	2.0167
7	2.4558	2.1914

(c)

Figure 3.3: Value of conditional entropy.

distinguish them. Figure 3.3 (c) shows the values of $\text{INF}^k_{I_1}(p \mid \Sigma_3)$ and $\text{INF}^k_{I_2}(p \mid \Sigma_3)$ for $k = 5, 6, 7$. As expected, the values are smaller for I_2. In fact, $\text{INF}_{I_1}(p \mid \Sigma_3) = 0.875$ and $\text{INF}_{I_2}(p \mid \Sigma_3) = 0.78125$. $\square$

3.4.1 Basic Properties

It is clear from the definitions that $\text{INF}_I(p \mid \Sigma)$ does not depend on a particular enumeration of positions. Two other basic properties that we can expect from the measure of information content are as follows: first, it should not depend on a particular representation of constraints, and second, a schema without constraints must be well-designed (as there is nothing to tell us that it is not). Both are indeed true.

Proposition 3.4.8

1) *Let Σ_1 and Σ_2 be two sets of constraints over a schema S. If they are equivalent (that is, $\Sigma_1^+ = \Sigma_2^+$), then for any instance I satisfying Σ_1 and any $p \in Pos(I)$, $\text{INF}_I(p \mid \Sigma_1) = \text{INF}_I(p \mid \Sigma_2)$.*

2) *If $\Sigma = \emptyset$, then (S, Σ) is well-designed.*

PROOF:

1) Follows from the fact that for every instance I of S, $I \models \Sigma_1$ iff $I \models \Sigma_2$. Hence, for every $a \in [1, k]$ and $\bar{a} \in \Omega(I, p)$, $SAT^k_{\Sigma_1}(I_{(a,\bar{a})}) = SAT^k_{\Sigma_2}(I_{(a,\bar{a})})$ and, therefore, $H(\mathcal{B}^k_{\Sigma_1}(I, p) \mid \mathcal{A}(I, p)) = H(\mathcal{B}^k_{\Sigma_2}(I, p) \mid \mathcal{A}(I, p))$.

2) Follows from part 2) of Proposition 3.4.9, to be proved below. Since for every $I \in inst(S, \Sigma)$, $p \in Pos(I)$ and $a \in \mathbb{N}^+ - adom(I)$, we have $I_{p \leftarrow a} \models \Sigma$, this implies that (S, Σ) is well-designed.

□

In the following proposition we show a very useful structural criterion for $\textsc{Inf}_I(p \mid \Sigma) = 1$, namely that a schema (S, Σ) is well-designed if and only if one position of an arbitrary $I \in inst(S, \Sigma)$ can always be assigned a fresh value. Also in this proposition, we use this criterion to show that $\textsc{Inf}^k_I(p \mid \Sigma)$ cannot exhibit sub-logarithmic growth, that is, if $\lim_{k\to\infty} \textsc{Inf}^k_I(p \mid \Sigma)/\log k = 1$, then $\lim_{k\to\infty}[\log k - \textsc{Inf}^k_I(p \mid \Sigma)] = 0$.

Proposition 3.4.9 *Let S be a schema and Σ a set of constraints over S. Then the following are equivalent.*

1) (S, Σ) is well-designed.

2) For every $I \in inst(S, \Sigma)$, $p \in Pos(I)$ and $a \in \mathbb{N}^+ - adom(I)$, $I_{p\leftarrow a} \models \Sigma$.

3) For every $I \in inst(S, \Sigma)$ and $p \in Pos(I)$, $\lim_{k\to\infty}[\log k - \textsc{Inf}^k_I(p \mid \Sigma)] = 0$.

This proposition shows that the information-theoretic definition of being well-designed is equivalent to the set-theoretic definition presented in Section 2.1.4, and used by Vincent [Vin99] to show that 4NF precisely characterizing redundancy in relational databases containing functional and multivalued dependencies. We use here an entropy-based approach because we would also like to measure the amount of redundancy in a relational database and, in particular, we would like to reason about normalization algorithms and show that the usual decomposition step in these algorithms reduces the amount of redundancy in a database. We cannot possibly do this with the set-theoretic measure because it is too coarse. In particular, given that the set-theoretic approach does not measure the amount of redundancy, it cannot distinguish between instances containing redundant information and cannot distinguish between different types of dependencies that can cause different degrees of redundancy (think of a dependency that says that all the values in a database must be the same).

In Section 3.4.2, we use our information-theoretic approach to justify normal forms such as BCNF, 4NF, PJ/NF, 5NFR and DK/NF. It should be noted that we can also use the set-theoretic measure to justify these normal forms. The advantage of our measure is that it can also be used to reason about normal forms that allow redundant information, such as 3NF [Kol05], and to reason about databases containing attributes with finite domains, where the set-theoretic approach cannot be directly applied.

The following lemma will be used in the proof of Proposition 3.4.9 and in several other proofs.

Lemma 3.4.10 *Fix $n, m > 0$, an n-element set A and a probability space $\mathcal{A}$ on A with the uniform distribution $P_{\mathcal{A}}$. Assume that for each $k > 0$, we have a probability space on $[1, k]$ called $\mathcal{B}_k$ and a joint distribution $P_{\mathcal{B}_k,\mathcal{A}}$ on $[1, k] \times A$ such that for some $a_0 \in A$, and for all $k > 0$, the conditional probability $P(i \mid a_0) = P_{\mathcal{B}_k,\mathcal{A}}(i, a_0)/P_{\mathcal{A}}(a_0) = 0$, for at least $k - m$ elements of $[1, k]$. Then for every $k > m^2$:*

$$\frac{H(\mathcal{B}_k \mid \mathcal{A})}{\log k} \;<\; 1 - \frac{1}{2n}.$$

In particular, if $\lim_{k\to\infty} H(\mathcal{B}_k \mid \mathcal{A})/\log k$ exists, then $\lim_{k\to\infty} H(\mathcal{B}_k \mid \mathcal{A})/\log k \;<\; 1$.

PROOF: First, assume that $m > 1$. Let $k > m^2$ and $M = \{i \in [1, k] \mid P(i \mid a_0) > 0\}$. Observe that $|M| \leq m$. Then $\frac{H(\mathcal{B}_k|\mathcal{A})}{\log k}$ is equal to

$$\begin{aligned}
& \frac{1}{\log k}\left[\sum_{a\in A}\frac{1}{n}\sum_{i\in[1,k]} P(i \mid a)\log\frac{1}{P(i \mid a)}\right] \\
= \;& \frac{1}{n\log k}\left[\left(\sum_{a\in A-\{a_0\}}\sum_{i\in[1,k]} P(i \mid a)\log\frac{1}{P(i \mid a)}\right)+\left(\sum_{i\in[1,k]} P(i \mid a_0)\log\frac{1}{P(i \mid a_0)}\right)\right] \\
= \;& \frac{1}{n\log k}\left[\left(\sum_{a\in A-\{a_0\}}\sum_{i\in[1,k]} P(i \mid a)\log\frac{1}{P(i \mid a)}\right)+\left(\sum_{i\in M} P(i \mid a_0)\log\frac{1}{P(i \mid a_0)}\right)\right] \\
\leq \;& \frac{1}{n\log k}\left[\left(\sum_{a\in A-\{a_0\}}\log k\right)+\log m\right] \qquad (3.1) \\
= \;& \frac{1}{n\log k}\Big[(n-1)\log k+\log m\Big] \\
= \;& 1-\frac{1}{n}+\frac{\log m}{n\log k} \;<\; 1-\frac{1}{n}+\frac{\log m}{n\log m^2} \;=\; 1-\frac{1}{n}+\frac{1}{2n} \;=\; 1-\frac{1}{2n}.
\end{aligned}$$

Now, assume that $m = 1$. In this case, $\log m$ in equation (3.1) is equal to 0 and, therefore, the previous sequence of formulas show that $H(\mathcal{B}_k \mid \mathcal{A})/\log k \;\leq\; 1-\frac{1}{n} \;<\; 1-\frac{1}{2n}$. $\square$

PROOF OF PROPOSITION 3.4.9: We will prove the chain of implications 3) $\Rightarrow$ 1) $\Rightarrow$ 2) $\Rightarrow$ 3).

The implication 3) $\Rightarrow$ 1) is straightforward. Next we show 1) $\Rightarrow$ 2). Towards a contradiction, assume that there exists $I \in \mathit{inst}(S, \Sigma)$, $p \in \mathit{Pos}(I)$ and $a \in \mathbb{N}^+ - \mathit{adom}(I)$ such that $I_{p\leftarrow a} \not\models \Sigma$. Let $k > 0$ be such that $\mathit{adom}(I) \cup \{a\} \subseteq [1, k]$. By Claim A.1.1 (see Appendix), for every $b \in [1, k] - \mathit{adom}(I)$, $I_{p\leftarrow b} \not\models \Sigma$. Thus, for every $a \in [1, k] - \mathit{adom}(I)$, $P(a \mid \bar{a}_0) = 0$, where $\bar{a}_0$ is the tuple in $\Omega(I, p)$ containing no variables. Therefore, applying

Lemma 3.4.10 with $n = 2^{\|I\|-1}$ and $m = |adom(I)|$, we conclude that for $k > m^2$:

$$\frac{\text{Inf}_I^k(p \mid \Sigma)}{\log k} \;=\; \frac{H(\mathcal{B}_\Sigma^k(I,p) \mid \mathcal{A}(I,p))}{\log k} \;<\; 1 - \frac{1}{2 \cdot 2^{\|I\|-1}}.$$

Since $\text{Inf}_I(p \mid \Sigma) = \lim_{k\to\infty} \text{Inf}_I^k(p \mid \Sigma)/\log k$ exists by Lemma 3.4.4, we conclude that $\text{Inf}_I(p \mid \Sigma) < 1$ and thus (S, Σ) is not well-designed, a contradiction.

Next, we show 2) $\Rightarrow$ 3). Let $I \in inst(I, \Sigma)$ and $p \in Pos(I)$. Let $\|I\| = n$ and let $k > n$ be such that $I \in inst_k(S, \Sigma)$. First, we prove that for every $a \in [1,k] - adom(I)$ and $\bar{a} \in \Omega(I,p)$,

$$|SAT_\Sigma^k(I_{(a,\bar{a})})| \;\geq\; (k-n)^{|var(\bar{a})|} \tag{3.2}$$

where $var(\bar{a})$ is the set of variables in $\bar{a}$. We do it by induction on $|var(\bar{a})|$ [3]. We do it by induction on $|var(\bar{a})|$. Assume that $|var(\bar{a})| = 0$. Then given that $I_{p\leftarrow a} \models \Sigma$, we conclude that $|SAT_\Sigma^k(I_{(a,\bar{a})})| = 1$. Now assume that (3.2) is true for every tuple in $\Omega(I,p)$ containing at most m variables, and let $|var(\bar{a})| = m+1$. Suppose that $\bar{a} = (a_1, \ldots, a_{p-1}, a_{p+1}, \ldots, a_n)$ and $a_i = v_i$, for some $i \in [1, p-1] \cup [p+1, n]$. Let $I' = I_{p\leftarrow a}$. By the assumption, $I' \models \Sigma$, and hence for every $b \in [1,k] - adom(I')$ we have $I'_{i\leftarrow b} \models \Sigma$. Thus, given that $|[1,k] - adom(I')| \geq k-n$ and for every $b_1, b_2 \in [1,k] - adom(I')$, $|SAT_\Sigma^k(I'_{(a,\bar{b}_1)})| = |SAT_\Sigma^k(I'_{(a,\bar{b}_2)})|$, where $\bar{b}_j$ $(j = 1,2)$ is a tuple constructed from $\bar{a}$ by replacing v_i by b_j, we conclude that if $\bar{b}$ is a tuple constructed from $\bar{a}$ by replacing v_i by an arbitrary $b \in [1,k] - adom(I')$, then $|SAT_\Sigma^k(I_{(a,\bar{a})})| \geq (k-n) \cdot |SAT_\Sigma^k(I'_{(a,\bar{b})})|$, since $|adom(I')| \leq n$. By the induction hypothesis, $|SAT_\Sigma^k(I'_{(a,\bar{b})})| \geq (k-n)^{|var(\bar{b})|} = (k-n)^{|var(\bar{a})|-1}$ and, therefore, $|SAT_\Sigma^k(I_{(a,\bar{a})})| \geq (k-n)^{|var(\bar{a})|}$, proving (3.2).

Now we show that $\lim_{k\to\infty}[\log k - \text{Inf}_I^k(p \mid \Sigma)] = 0$. For every $k \geq 1$ such that $adom(I) \subseteq [1,k]$, $\log k \geq \text{Inf}_I^k(p \mid \Sigma)$ and, therefore, $\lim_{k\to\infty}[\log k - \text{Inf}_I^k(p \mid \Sigma)] \geq 0$. Hence, to prove the theorem we will show that

$$\lim_{k\to\infty}[\log k - \text{Inf}_I^k(p \mid \Sigma)] \;\leq\; 0. \tag{3.3}$$

Let $k \geq 1$ be such that $adom(I) \subseteq [1,k]$. Assume that $k > n$. Let $a \in [1,k] - adom(I)$ and $\bar{a} \in \Omega(I,p)$. Since $\sum_{b\in[1,k]} |SAT_\Sigma^k(I_{(b,\bar{a})})| \leq k^{|var(\bar{a})|+1}$, using (3.2), we get

$$P(a \mid \bar{a}) \;\geq\; \frac{(k-n)^{|var(\bar{a})|}}{k^{|var(\bar{a})|+1}} \;=\; \frac{1}{k}\,(1 - \frac{n}{k})^{|var(\bar{a})|}. \tag{3.4}$$

[3] This induction relies on the following simple idea: If $a \notin adom(I)$, then $I_{p\leftarrow a} \models \Sigma$ and, therefore, one can replace values in positions of $\bar{a}$ one by one, provided that each position gets a fresh value.

By Claim A.1.1 (see Appendix), for every $a, b \in [1,k] - adom(I)$ and every $\bar{a} \in \Omega(I,p)$, $P(a \mid \bar{a}) = P(b \mid \bar{a})$. Thus, for every $a \in [1,k] - adom(I)$ and every $\bar{a} \in \Omega(I,p)$,

$$P(a \mid \bar{a}) \;\leq\; 1/(k - |adom(I)|) \;\leq\; 1/(k-n). \tag{3.5}$$

In order to prove (3.3), we need to establish a lower bound for $\text{INF}^k_I(p \mid \Sigma)$. We do this by using (3.4) and (3.5) as follows: Given the term $P(a \mid \bar{a}) \log \frac{1}{P(a|\bar{a})}$, we use (3.4) and (3.5) to replace $P(a \mid \bar{a})$ and $\log \frac{1}{P(a|\bar{a})}$ by smaller terms, respectively. More precisely,

$$\begin{aligned}
\text{INF}^k_I(p \mid \Sigma) &= \sum_{\bar{a} \in \Omega(I,p)} \left(P(\bar{a}) \sum_{a \in [1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \right) \\
&\geq \frac{1}{2^{n-1}} \sum_{a \in [1,k]-adom(I)} \; \sum_{\bar{a} \in \Omega(I,p)} \frac{1}{k} \, (1 - \frac{n}{k})^{|var(\bar{a})|} \log(k-n) \\
&= \frac{1}{2^{n-1}} \; \log(k-n) \; \frac{1}{k} \sum_{a \in [1,k]-adom(I)} \; \sum_{i=0}^{n-1} \binom{n-1}{i} (1 - \frac{n}{k})^i \\
&= \frac{1}{2^{n-1}} \; \log(k-n) \; \frac{1}{k} \sum_{a \in [1,k]-adom(I)} ((1 - \frac{n}{k}) + 1)^{n-1} \\
&\geq \frac{1}{2^{n-1}} \; \log(k-n) \; \frac{1}{k} \; (k-n) \; (2 - \frac{n}{k})^{n-1} \\
&\geq \frac{1}{2^{n-1}} \; \log(k-n) \; \frac{1}{k} \; (k-n) \; (2 - \frac{2n}{k})^{n-1} \\
&= \frac{1}{2^{n-1}} \; \log(k-n) \; (1 - \frac{n}{k}) \; 2^{n-1} \; (1 - \frac{n}{k})^{n-1} \\
&= \log(k-n) \; (1 - \frac{n}{k})^n.
\end{aligned}$$

Therefore, $\log k - \text{INF}^k_I(p \mid \Sigma) \leq \log k - \log(k-n) \; (1 - \frac{n}{k})^n$. Since $\lim_{k \to \infty}[\log k - \log(k - n) \; (1 - \frac{n}{k})^n] = 0$ we conclude that (3.3) holds. This completes the proof of Proposition 3.4.9. $\square$

A natural question at this point is whether the problem of checking if a relational schema is well-designed is decidable. It is not surprising that for arbitrary first-order constraints, the problem is undecidable:

Proposition 3.4.11 *The problem of verifying whether a relational schema containing first-order constraints is well-designed is undecidable.*

PROOF: It is known that the problem of verifying whether a first-order sentence φ of the form $\exists\bar{x}\forall\bar{y}\, \psi(\bar{x}, \bar{y})$, where $\psi(\bar{x}, \bar{y})$ is an arbitrary first-order formula, is finitely satisfiable is undecidable. Denote this decision problem by $\mathcal{P}_{\exists\forall}$.

We will reduce $\mathcal{P}_{\exists\forall}$ to the complement of our problem. Let φ be a formula of the form shown above. Assume that φ is defined over a relational schema $\{R_1, \ldots, R_n\}$ and $|\bar{x}| = m > 0$, and let S be a relational schema $\{U_1, U_2, R_1, \ldots, R_n\}$, where U_1, U_2 are m-ary predicates. Furthermore, define a set of constraints Σ over S as follows:

$$\Sigma \quad = \quad \{\forall \bar{x}\, (U_1(\bar{x}) \leftrightarrow U_2(\bar{x})),\ \forall \bar{x}\, (U_1(\bar{x}) \rightarrow \forall \bar{y}\, \psi(\bar{x}, \bar{y}))\}. \tag{3.6}$$

It suffices to show that $\varphi \in \mathcal{P}_{\exists\forall}$ if and only if (S, Σ) is not well-designed.

($\Rightarrow$) Assume that $\varphi \in \mathcal{P}_{\exists\forall}$ and let I_0 be an instance of $\{R_1, \ldots, R_n\}$ satisfying φ. Define $I \in \mathit{inst}(S, \Sigma)$ as follows: $I(R_i) = I_0(R_i)$, for every $i \in [1, n]$, and $I(U_1) = I(U_2) = \{\bar{a}\}$, where $\bar{a}$ is an m-tuple in I_0 such that $I_0 \models \forall \bar{y}\, \psi(\bar{a}, \bar{y})$. Let $a \in \mathbb{N}^+ - \mathit{adom}(I)$ and p be an arbitrary position in $I(U_1)$. Then $I_{p \leftarrow a} \not\models \forall \bar{x}\, (U_1(\bar{x}) \leftrightarrow U_2(\bar{x}))$ and, therefore, (S, Σ) is not well-designed by Proposition 3.4.9.

($\Leftarrow$) Assume that $\varphi \notin \mathcal{P}_{\exists\forall}$. Then for every nonempty instance $I \in \mathit{inst}(S, \Sigma)$, $I(U_1) = I(U_2) = \emptyset$ and $I(R_i) \neq \emptyset$, for some $i \in [1, n]$. But for every position p of a value in $I(R_j)$ ($j \in [1, n]$) and every $a \in \mathbb{N}^+ - \mathit{adom}(I)$, $I_{p \leftarrow a} \models \Sigma$ since $I(U_1)$ and $I(U_2)$ are empty. We conclude that (S, Σ) is well-designed by Proposition 3.4.9. $\square$

However, integrity constraints used in database schema design are most commonly *universal*, that is, of the form $\forall \bar{x}\, \psi(\bar{x})$, where $\psi(\bar{x})$ is a quantifier-free formula. FDs, MVDs and JDs are universal constraints as well as more elaborated dependencies such as equality-generating dependencies and full tuple-generating dependencies [AHV95]. For universal constraints, the problem of testing if a relational schema is well-designed is decidable. In fact,

Proposition 3.4.12 *The problem of deciding whether a schema containing only universal constraints is well-designed is co-NEXPTIME-complete. Furthermore, if for a fixed m, each relation in S has at most m attributes, then the problem is Π_2^p-complete.*

To prove this proposition, first we have to prove a lemma. In this lemma we use the following terminology. A first-order constraint φ is a Σ_n-*sentence* if φ is of the form $Q_1 x_1 Q_2 x_2 \cdots Q_m x_m \psi$, where (1) $Q_i \in \{\forall, \exists\}$ ($i \in [1, m]$); (2) ψ is a quantifier-free formula; (3) the string of quantifiers $Q_1 Q_2 \cdots Q_m$ consists of n consecutive blocks, all quantifiers in the same block are the same and no adjacent blocks have the same quantifiers; and (4) the first block contains existential quantifiers. Moreover, Π_n-sentences are defined analogously, but requiring that the first block contains universal quantifiers.

Lemma 3.4.13 *Let S be a relational schema and Σ be a set of $\Sigma_n \cup \Pi_n$-sentences over S, $n \geq 1$. Then there exists a relational schema $S' \supseteq S$ and a Π_{n+1}-sentence φ over S' such that (S, Σ) is well-designed iff $\varphi \in \Sigma^+$. Moreover, $\|\varphi\|$ is $O(\|(S,\Sigma)\|^2)$.*

PROOF: Assume that $S = \{R_1^{m_1}, \ldots, R_n^{m_n}\}$, where m_i is the arity of R_i ($i \in [1,n]$). Define a relational schema S' as $S \cup \{R_{i,j}^{m_i} \mid i \in [1,n] \text{ and } j \in [1,m_i]\} \cup \{U^1\}$. To define φ, first we define sentence ψ as the conjunction of the following formulas.

- $\bigvee_{i=1}^n \exists x_1 \cdots \exists x_{m_i} R_i(x_1, \ldots, x_{m_i})$. For some $i \in [1,n]$, relation R_i is not empty.
- $\exists x\,(U(x) \wedge \forall y\,(U(y) \rightarrow x = y))$. U contains exactly one element.
- For every $i \in [1,n]$,

$$\forall x \forall y_1 \cdots \forall y_{m_i-1}\,(U(x) \rightarrow \bigwedge_{j=1}^{m_i} \neg R_i(y_1, \ldots, y_{j-1}, x, y_j, \ldots, y_{m_i-1})).$$

  That is, the element contained in U is not contained in the active domain of relation R_i, for every $i \in [1,n]$.

- For every $i \in [1,n]$,

$$(\forall x_1 \cdots \forall x_{m_i} \neg R_i(x_1, \ldots, x_{m_i})) \rightarrow (\bigwedge_{j=1}^{m_i} \forall y_1 \cdots \forall y_{m_i} \neg R_{i,j}(y_1, \ldots, y_{m_i})).$$

  If R_i is empty, then $R_{i,j}$ is empty, for every $j \in [1, m_i]$.

- For every $i \in [1,n]$ and every $j \in [1,m_i]$,

$$\begin{aligned}
&\exists u_1 \cdots \exists u_{m_i} R_i(u_1, \ldots, u_{m_i}) \rightarrow \\
&\quad \exists x \exists x' \exists y_1 \cdots \exists y_{j-1} \exists y_{j+1} \cdots \exists y_{m_i}\,(\\
&\qquad R_i(y_1, \ldots, y_{j-1}, x, y_{j+1}, \ldots, y_{m_i}) \wedge \\
&\qquad \neg R_{i,j}(y_1, \ldots, y_{j-1}, x, y_{j+1}, \ldots, y_{m_i}) \wedge \\
&\qquad R_{i,j}(y_1, \ldots, y_{j-1}, x', y_{j+1}, \ldots, y_{m_i}) \wedge U(x') \wedge \\
&\qquad \forall z_1 \cdots \forall z_{m_i}\,((z_j \neq x \wedge z_j \neq x') \vee \bigvee_{k=1, k\neq j}^{m_i} z_k \neq y_k \rightarrow \\
&\qquad\qquad (R_i(z_1, \ldots, z_{m_i}) \leftrightarrow R_{i,j}(z_1, \ldots, z_{m_i})))).
\end{aligned}$$

  If R_i is not empty, then there exists a tuple t in R_i and a tuple t' in $R_{i,j}$ such that t' is not in R_i, t is not in $R_{i,j}$ and t, t' contain exactly the same values, except for the element in the j-th column where t' contains a value that is in relation U. Furthermore, every other tuple is in R_i if and only if is in $R_{i,j}$.

Given $i \in [1, n]$ and $j \in [1, m_i]$, we denote by $\Sigma[R_i/R_{i,j}]$ the set of first-order constraints generated from Σ by replacing every occurrence of R_i by $R_{i,j}$. We define sentence φ as follows:

$$\psi \rightarrow \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{m_i} \Sigma[R_i/R_{i,j}]. \tag{3.7}$$

Notice that ψ is a Σ_2-sentence and, therefore, φ is a Π_{n+1}-sentence, since $n \geq 1$. To finish the proof, we have to show that (S, Σ) is well-designed if and only if $\varphi \in \Sigma^+$.

($\Leftarrow$) Assume that (S, Σ) is not well-designed. Then by Proposition 3.4.9, there exists $I \in \mathit{inst}(S, \Sigma)$, $p \in \mathit{Pos}(I)$ and $a \in \mathbb{N}^+ - \mathit{adom}(I)$ such that $I_{p \leftarrow a} \not\models \Sigma$. Assume that p is the position of some element in the j_0-th column of R_{i_0} ($i_0 \in [1, n]$, $j_0 \in [1, m_{i_0}]$). Then we define an instance I' of S' as follows. For every $i \in [1, n]$, $I'(R_i) = I(R_i)$, $I(U) = \{a\}$ and $I'(R_{i_0,j_0}) = I_{p \leftarrow a}(R_{i_0})$. Furthermore, for every $i \in [1, n]$ and $j \in [1, m_i]$, with $i \neq i_0$ or $j \neq j_0$, if $I(R_i)$ is empty, then $I'(R_{i,j})$ is also empty, else $I'(R_{i,j})$ is constructed by replacing an arbitrary element in the j-th column of $I(R_i)$ by a. Then $I' \models \Sigma$, since $I \models \Sigma$ and $I'(R_i) = I(R_i)$ for every $i \in [1, n]$. $I' \models \psi$ since (1) $I'(R_{i_0})$ is not empty ($I(R_{i_0})$ is not empty); (2) $I'(U) = \{a\}$ and $a \notin \mathit{adom}(I)$; (3) for every $i \in [1, n]$, if $I'(R_i)$ is empty, then $I'(R_{i,j})$ is empty, for every $j \in [1, m_i]$; and (4) for every $i \in [1, n]$, $j \in [1, m_i]$, if $I'(R_i)$ is not empty, then $I'(R_{i,j})$ differs from $I'(R_i)$ by exactly one value, which is in U. Finally, $I' \not\models \Sigma[R_{i_0}/R_{i_0,j_0}]$, since $I'(R_{i_0,j_0}) = I_{p \leftarrow a}(R_{i_0})$ and $I_{p \leftarrow a} \not\models \Sigma$. We conclude that $I' \not\models \varphi$ and, therefore, $\varphi \notin \Sigma^+$.

($\Rightarrow$) Assume that $\varphi \notin \Sigma^+$. Then there exists a database instance I' of S', $i_0 \in [1, n]$ and $j_0 \in [1, m_{i_0}]$ such that $I' \models \Sigma$, $I' \models \psi$ and $I' \not\models \Sigma[R_{i_0}/R_{i_0,j_0}]$. We note that $I'(R_{i_0})$ is not empty (if $I'(R_{i_0})$ is empty, then $I'(R_{i_0,j_0})$ is empty ($I' \models \psi$) and, therefore, $I'(R_{i_0,j_0}) = I'(R_{i_0})$ and $I' \models \Sigma[R_{i_0}/R_{i_0,j_0}]$, since $I' \models \Sigma$, a contradiction). Define an instance I of S as follows. For every $i \in [1, n]$, $I(R_i) = I'(R_i)$. Let a be the element in $I'(U)$ and let p be the position in I of the element that has to be changed to obtain $I'(R_{i_0,j_0})$ from $I(R_{i_0})$. Then (1) I is not empty, since $I' \models \psi$; (2) $I \models \Sigma$, since $I' \models \Sigma$ and $I(R_i) = I'(R_i)$, for every $i \in [1, n]$; and (3) $I_{p \leftarrow a} \not\models \Sigma$, since $I' \not\models \Sigma[R_{i_0}/R_{i_0,j_0}]$. Given that $a \in \mathbb{N}^+ - \mathit{adom}(I)$, since $I' \models \psi$, by Proposition 3.4.9 we conclude that (S, Σ) is not well-designed. □

Σ_2-sentences correspond to the Schönfinkel-Bernays fragment of first-order logic. It is known that the problem of verifying if a Schönfinkel-Bernays formula has a finite model is NEXPTIME-complete [Pap94] and becomes Σ_2^p-complete if every relation has at most m attributes, where m is a fixed constant. Thus, from Lemma 3.4.13 we obtain the following corollary and the proof of Proposition 3.4.12.

Corollary 3.4.14 *The problem of deciding whether a schema containing only $\Sigma_1 \cup \Pi_1$-sentences is well-designed belongs to co-NEXPTIME.*

PROOF OF PROPOSITION 3.4.12: We consider only the case of unbounded-arity relations, being the case of fixed-arity relations similar. The membership part of the proposition is a particular case of Corollary 3.4.14. The hardness part of the proposition follows from the following observation. If in the reduction of Proposition 3.4.11 the formula φ is of the form $\exists \bar{x} \forall \bar{y}\, \psi(\bar{x}, \bar{y})$, where ψ is quantifier-free, then the set of constraints Σ defined in (3.6) is universal. Thus, the same reduction of Proposition 3.4.11 shows that the problem of deciding whether a Σ_2-sentence is finitely satisfiable is reducible to the problem of deciding whether a schema containing only universal constraints is well-designed. $\square$

For specific kinds of constraints, e.g., FDs, MVDs, lower complexity bounds will follow from the results in the next section.

3.4.2 Justification of Relational Normal Forms

We now apply the criterion of being well-designed to various relational normal forms. We show that all of them lead to well-designed specifications, and some precisely characterize the well-designed specifications that can be obtained with a class of constraints.

We start by finding constraints that always give rise to well-designed schemas. Recall that a *typed unirelational equality-generating dependency* [AHV95] is a constraint of the form:

$$\forall \; (R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow \bar{x} = \bar{y}),$$

where $\forall$ represents the universal closure of a formula, $\bar{x}, \bar{y} \subseteq \bar{x}_1 \cup \cdots \cup \bar{x}_m$ and there is an assignment of variables to columns such that each variable occurs only in one column and each equality atom involves a pair of variables assigned to the same column. An *extended key* is a typed unirelational equality-generating dependency of the form:

$$\forall\ (R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow \bar{x}_i = \bar{x}_j),$$

where $i, j \in [1, m]$. Note that every key is an extended key.

Proposition 3.4.15 *If S is a schema and Σ a set of extended keys over S, then (S, Σ) is well-designed.*

Before proving this proposition we introduce one definition that will be used in several proofs. Let $I \in \mathit{inst}(S, \Sigma)$, $p \in \mathit{Pos}(I)$, $a \in [1, k]$ and $\bar{a} \in \Omega(I, p)$. Given a substitution $\sigma : \bar{a} \rightarrow [1, k]$ and $R \in S$, we say that a tuple $t' \in \sigma(I_{(a,\bar{a})})(R)$ is *generated* by a tuple $t \in I(R)$ by means of a tuple $t^* \in I_{(a,\bar{a})}$ if $\sigma(t^*) = t'$ and t^* can be obtained from t by replacing each value in it by the element of $(a, \bar{a})$ in the same position. We say $t' \in \sigma(I_{(a,\bar{a})})(R)$ is generated by a tuple $t \in I(R)$ if it is generated by t by means of some $t^* \in I_{(a,\bar{a})}$.

PROOF OF PROPOSITION 3.4.15: To prove the proposition, we now use part 2) of Proposition 3.4.9. Let $I \in \mathit{inst}(S, \Sigma)$, $p \in \mathit{Pos}(I)$ and $a \in \mathbb{N}^+ - \mathit{adom}(I)$. We have to show that $I_{p \leftarrow a} \models \Sigma$.

Assume to the contrary that $I_{p \leftarrow a} \not\models \Sigma$. Then there exists $R \in S$ and an extended key $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow \bar{x}_i = \bar{x}_j) \in \Sigma$ such that $I_{p \leftarrow a} \not\models \forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow \bar{x}_i = \bar{x}_j)$. Thus, there exists a substitution $\rho' : \bar{x}_1 \cup \cdots \cup \bar{x}_m \rightarrow [1, k]$ such that $\rho'(\bar{x}_l) = t'_l$ and $t'_l \in I_{p \leftarrow a}(R)$, for every $l \in [1, m]$, and $t'_i \neq t'_j$. Define a substitution $\rho : \bar{x}_1 \cup \cdots \cup \bar{x}_m \rightarrow [1, k]$ as follows. Let b be the value in the p-th position of I. Then

$$\rho(x) = \begin{cases} \rho'(x) & \rho'(x) \neq a \\ b & \text{Otherwise} \end{cases}$$

Let $\rho(\bar{x}_l) = t_l$, for every $l \in [1, n]$. It is straightforward to verify that $t'_1, \ldots, t'_n$ are generated from $t_1, \ldots, t_n$, respectively. Given that $I \models \Sigma$, $t_i = t_j$ and, therefore, $t'_i = t'_j$. This contradiction proves the proposition. $\square$

From Section 2.1.2, recall that if (S, Σ) is a database schema such that Σ does not contain any domain dependency, then (S, Σ) is in DK/NF if and only if Σ is implied by the set of key dependencies in Σ^+. Thus, in our setting, where domain dependencies are not considered, we obtain the following corollary from Proposition 3.4.15.

Corollary 3.4.16 *A relational specification* (S, Σ) *in DK/NF is well-designed.*

In the rest of this section, we also denote join dependencies by first-order sentences. More precisely, a join dependency over a relation R is a first-order sentence of the form:

$$\forall\,(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \to R(\bar{x})),$$

where $\forall$ represents the universal closure of a formula, $\bar{x} \subseteq \bar{x}_1 \cup \cdots \cup \bar{x}_m$, every variable not in $\bar{x}$ occurs in precisely one $\bar{x}_i$ ($i \in [1, m]$) and there is an assignment of variables to columns such that each variable occurs only in one column. For example, join dependency $\bowtie[AB, BC]$ over a relation $R(A, B, C)$ can be denoted by

$$\forall x \forall y \forall z \forall u_1 \forall u_2\,(R(x, y, u_1) \wedge R(u_2, y, z) \to R(x, y, z)).$$

Next, we characterize well-designed schemas with FDs and JDs.

Theorem 3.4.17 *Let* Σ *be a set of FDs and JDs over a relational schema* S. (S, Σ) *is well-designed if and only if for every* $R \in S$ *and every nontrivial join dependency* $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \to R(\bar{x}))$ *in* Σ^+*, there exists* $M \subseteq \{1, \ldots, m\}$ *such that:*

1) $\bar{x} \subseteq \bigcup_{i \in M} \bar{x}_i$.

2) For every $i, j \in M$, $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \to \bar{x}_i = \bar{x}_j) \in \Sigma^+$.

In the proof of Theorem 3.4.17 we shall use chase for FDs and JDs [MMS79] which was introduced in Section 2.1. Chase can be generalized in a natural manner to the case of more expressive constraints like typed equality-generating dependencies (see [AHV95]).

We now move to the proof of Theorem 3.4.17. We need two lemmas first.

Lemma 3.4.18 *Let* Σ *be a set of FDs and JDs over a relational schema* S *and* $R \in S$. *Assume that* Σ *contains a JD* $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \to R(\bar{x}))$ *such that* $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \to \bar{x} = \bar{x}_i) \notin \Sigma^+$, *for every* $i \in [1, m]$. *Then there exists* $I \in inst(S, \Sigma)$ *and* $p \in Pos(I)$ *such that* $\textrm{INF}_I(p \mid \Sigma) < 1$.

Proof: Let T be a tableau containing tuples $\{\bar{x}_1, \ldots, \bar{x}_m\}$, and let $\bar{x}$ be the distinguished variables. Let ρ be a one-to-one function with the domain $\bar{x}_1 \cup \cdots \cup \bar{x}_m$ and the range contained in $\mathbb{N}^+$. Define $I = \rho(Chase_\Sigma(T))$. Assume that θ is the composition of the substitutions used in the chase. Let $t_j = \rho(\theta(\bar{x}_j))$, for every $j \in [1, m]$, and $t = \rho(\theta(\bar{x}))$. Given that $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \to \bar{x} = \bar{x}_i) \notin \Sigma^+$, for every $i \in [1, m]$, we conclude that $t \neq t_j$, for every $j \in [1,m]$. Let $A \in \mathit{sort}(R)$, p be the position of $t[A]$ in I and k such that $\mathit{adom}(I) \subseteq [1,k]$. Since $I \models \Sigma$ and I contains $t_1, \ldots, t_m$, the JD $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow R(\bar{x})) \in \Sigma$ implies that I must contain t. Thus, changing any value in t generates an instance that does not satisfy Σ. Hence, for every $a \in [1,k] - \{t[A]\}$, $P(a \mid \bar{a}_0) = 0$, where $\bar{a}_0$ is the tuple in $\Omega(I,p)$ containing no variables. Applying Lemma 3.4.10 we conclude that $H(\mathcal{B}^k_\Sigma(I,p) \mid \mathcal{A}(I,p))/\log k < c$ for some constant $c < 1$, for all sufficiently large k, and thus by Lemma 3.4.4, $\textsc{Inf}_I(p \mid \Sigma) = \lim_{k\to\infty} \textsc{Inf}^k_I(p \mid \Sigma)/\log k < 1$. □

Given a set Σ of FDs and JDs over a relational schema S and a JD $\varphi \in \Sigma$ of the form $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow R(\bar{x}))$, define an equivalence relation $\sim_\varphi$ on tuples of variables as follows. For every $i,j \in [1,m]$, $\bar{x}_i \sim_\varphi \bar{x}_j$ if $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow \bar{x}_i = \bar{x}_j) \in \Sigma^+$. Let $[i]_\varphi$ be the equivalence class of $\bar{x}_i$, for every $i \in [1,m]$, and let $\mathit{var}([i]_\varphi)$ be the set of variables contained in all the tuples $\bar{x}_j \in [i]_\varphi$.

Lemma 3.4.19 *Let Σ be a set of FDs and JDs over a relational schema S and $R \in S$. Assume that Σ contains a JD φ of the form $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow R(\bar{x}))$ such that $\bar{x} \not\subseteq \mathit{var}([i]_\varphi)$, for every $i \in [1,m]$. Then there exists $I \in \mathit{inst}(S,\Sigma)$ and $p \in \mathit{Pos}(I)$ such that $\textsc{Inf}_I(p \mid \Sigma) < 1$.*

PROOF: If $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow \bar{x} = \bar{x}_i) \notin \Sigma^+$, for every $i \in [1,m]$, then by Lemma 3.4.18 there exists $I \in \mathit{inst}(S,\Sigma)$ and $p \in \mathit{Pos}(I)$ such that $\textsc{Inf}_I(p \mid \Sigma) < 1$. Thus, we may assume that there exists $i \in [1,m]$ such that $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow \bar{x} = \bar{x}_i) \in \Sigma^+$. By the hypothesis, there exists $l \in [1,|\bar{x}|]$ and a variable x in the l-th column of $\bar{x}$ such that $x \notin \mathit{var}([i]_\varphi)$. Let u be the variable in the l-th column of $\bar{x}_i$ and U_i the set of variables in the l-column of all the tuples $\bar{x}_j$ ($j \in [1,m]$) such that $\bar{x}_i \sim_\varphi \bar{x}_j$.

Let T be a tableau $\{\bar{x}_1, \ldots, \bar{x}_m\}$, with $\bar{x}_i$ as distinguished variables. In $\mathit{Chase}_\Sigma(T)$, all the tuples in the equivalence class of $\bar{x}_i$ (and no other) are identified with this tuple. Denote the l-th component of tuple $\bar{x}_j$ by $\bar{x}^l_j$ (and similarly for other tuples).

Let ρ be a one-to-one function with the domain $\bar{x}_1 \cup \cdots \cup \bar{x}_m$ and the range contained in $\mathbb{N}^+$ and $I = \rho(\mathit{Chase}_\Sigma(T))$. Assume that θ is the composition of the substitutions used in the chase. Let $t_j = \rho(\theta(\bar{x}_j))$ be a tuple in I, for every $j \in [1,m]$. Note that $\rho(\theta(\bar{x}_i)) = \rho(\bar{x}_i)$ since $\bar{x}_i$ is a tuple of distinguished variables. Additionally, since I satisfies $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow \bar{x} = \bar{x}_i)$, it must be the case that $\rho(\theta(\bar{x})) = \rho(\bar{x}_i)$.

Let p be the position in I of t_i^l. The value in this position is $\rho(u)$. We will show that for every $a \in [1,k] - \{\rho(u)\}$, $P(a \mid \bar{a}_0) = 0$, where $\bar{a}_0$ is a tuple in $\Omega(I,p)$ containing no variables.

Denote by t'_j the tuple of $I_{(a,\bar{a}_0)}$ that corresponds to t_j in I. Note that $t'_j = t_j$ for all j such that $\bar{x}_j$ is not in $[i]_\varphi$. When $\bar{x}_j$ is in $[i]_\varphi$, t'_j differs from t_j only in that the value in its l-th column is a rather than $\rho(u)$. Assume that $I_{(a,\bar{a}_0)}$ satisfies Σ. Then it satisfies, in particular, $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow R(\bar{x}))$. Recall that in this JD, every variable not in $\bar{x}$ occurs in a unique $\bar{x}_j$. We give a substitution from the variable tuples $\bar{x}_1, \ldots, \bar{x}_m$ to the tuples $t'_1, \ldots, t'_m$, respectively. Let $\rho' : \bar{x}_1 \cup \cdots \cup \bar{x}_m \rightarrow [1,k]$ be a substitution defined as follows. For every $y \in \bar{x}_1 \cup \cdots \cup \bar{x}_m$,

$$\rho'(y) = \begin{cases} \rho(\theta(y)) & \text{if } \; y \notin U_i \\ a & \text{otherwise.} \end{cases}$$

We claim that for every $j \in [1,m]$, $\rho'(\bar{x}_j) = t'_j$. Clearly, we only need to consider the l-th column. Indeed, if $\bar{x}_j$ is in $[i]_\varphi$, then t'_j is t_j, except in the l-column, where t_j contains the value a, since $\bar{x}_j^l$ is in U_i. Thus, $\rho'(\bar{x}_j) = t'_j$. If $\bar{x}_j$ is not in $[i]_\varphi$, then $\bar{x}_j^l$ is either x, or a variable that occurs only in $\bar{x}_j$. In either case, it is not in U_i. Thus, $\rho'(\bar{x}_j) = t'_j$. Since $I_{(a,\bar{a}_0)}$ is assumed to satisfy JD $\forall(R(\bar{x}_1) \wedge \cdots \wedge R(\bar{x}_m) \rightarrow R(\bar{x}))$, it must contain $\rho'(\bar{x})$. However, since x is not in U_i, $\rho'(\bar{x}) = \rho(\theta(\bar{x})) = \rho(\bar{x}_i) = t_i$ in I, which is not in $I_{(a,\bar{a}_0)}$, a contradiction.

We conclude that for every $a \in [1,k] - \{\rho(u)\}$, $P(a \mid \bar{a}_0) = 0$. Hence, by Lemma 3.4.10, $\text{INF}_I^k(p \mid \Sigma)/\log k < c$ for some constant $c < 1$, for all sufficiently large k, and then by Lemma 3.4.4, $\text{INF}_I(p \mid \Sigma) = \lim_{k\to\infty} \text{INF}_I^k(p \mid \Sigma)/\log k < 1$. This proves the lemma. $\square$

Theorem 3.4.17 is a corollary of Proposition 3.4.15 and Lemma 3.4.19. We note that this theorem justifies various normal forms proposed for JDs and FDs [Fag79, Vin97].

Corollary 3.4.20 *Let Σ be a set of FDs and JDs over a relational schema S. If (S,Σ) is in PJ/NF or 5NFR, then it is well-designed.*

However, neither of these normal forms characterizes precisely the notion of being well-designed:

Proposition 3.4.21 *There exists a schema S and a set of JDs and FDs Σ such that (S,Σ) is well-designed, but it violates all of the following: DK/NF, PJ/NF, 5NFR.*

PROOF: Let $S = \{R(A,B,C)\}$ and $\Sigma = \{AB \to C,\ AC \to B,\ \bowtie[AB,AC,BC]\}$. This specification is not in DK/NF and PJ/NF since the set of keys implied by Σ is $\{AB \to ABC, AC \to ABC, ABC \to ABC\}$ and this set does not imply $\bowtie[AB,AC,BC]$. Furthermore, this specification is not in 5NFR since $\bowtie[AB,AC,BC]$ is a strong-reduced join dependency and BC is not a key in Σ.

Join dependency $\bowtie[AB,AC,BC]$ corresponds to the following first order sentence:

$$\forall x \forall y \forall z \forall u_1 \forall u_2 \forall u_3\, (R(x,y,u_1) \wedge R(x,u_2,z)\ \wedge R(u_3,y,z) \to R(x,y,z)).$$

From Theorem 3.4.17, we conclude that (S,Σ) is well designed since Σ implies the sentence

$$\forall x \forall y \forall z \forall u_1 \forall u_2 \forall u_3 (R(x,y,u_1) \wedge R(x,u_2,z)\ \wedge R(u_3,y,z) \to y = u_2 \wedge z = u_1).$$

and $(x,y,z) \subseteq (x,y,u_1) \cup (x,u_2,z)$. $\square$

By restricting Theorem 3.4.17 to the case of specifications containing only FDs and MVDs (or only FDs), we obtain the equivalence between well-designed databases and 4NF (respectively, BCNF).

Theorem 3.4.22 *Let Σ be a set of integrity constraints over a relational schema S.*

1) If Σ contains only FDs and MVDs, then (S,Σ) is well-designed if and only if it is in 4NF.

2) If Σ contains only FDs, then (S,Σ) is well-designed if and only if it is in BCNF.

3.5 Normalization algorithms

We now show how the information-theoretic measure of Section 3.4 can be used for reasoning about normalization algorithms at the instance level. For this section, we assume that Σ is a set of FDs. The results shown here state that after each step of a decomposition algorithm, the amount of information in each position does not decrease.

Let I' be the result of applying one step of a normalization algorithm to I. In order to compare the amount of information in these instances, we need to show how to associate positions in I and I'. Since we only consider here functional dependencies, we deal with BCNF, and standard BCNF decomposition algorithms use steps of the following kind: pick a relation R with the set of attributes W, and let W be the disjoint union of X, Y, Z, such that $X \to Y \in \Sigma^+$. Then an instance $I = I(R)$ of R gets decomposed into $I_{XY} = \pi_{XY}(I)$ and $I_{XZ} = \pi_{XZ}(I)$, with the sets of FDs Σ_{XY} and Σ_{XZ}, where Σ_U stands for $\{C \to D \in \Sigma^+ \mid CD \subseteq U \subseteq W\}$. This decomposition gives rise to two partial maps $\pi_{XY} : Pos(I) \to Pos(I_{XY})$ and $\pi_{XZ} : Pos(I) \to Pos(I_{XZ})$. If p is the position of $t[A]$ for some $A \in XY$, then $\pi_{XY}(p)$ is defined, and equals the position of $\pi_{XY}(t)[A]$ in I_{XY}; the mapping π_{XZ} is defined analogously. Note that π_{XY} and π_{XZ} can map different positions in I to the same position of I_{XY} or I_{XZ}.

We now show that the amount of information in each position does not decrease in the normalization process.

Theorem 3.5.1 *Let (X, Y, Z) partition the attributes of R, and let $X \to Y \in \Sigma^+$. Let $I \in inst(R, \Sigma)$ and $p \in Pos(I)$. If U is either XY or XZ and π_U is defined on p, then* $\text{INF}_I(p \mid \Sigma) \;\leq\; \text{INF}_{I_U}(\pi_U(p) \mid \Sigma_U)$.

To prove this theorem, first we need to prove two lemmas.

Lemma 3.5.2 *Let Σ be a set of FDs over a relational schema S, $I \in inst(S, \Sigma)$, $p \in Pos(I)$ and $\bar{a} \in \Omega(I, p)$. Then $\lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a|\bar{a})}$ is either 0 or 1.*

PROOF: Given in Appendix A.2. □

Let R be a relation schema such that $sort(R) = X \cup Y \cup Z$, where X, Y and Z are nonempty pairwise disjoint sets of attributes. Let Σ be a set of FDs over R and $I \in inst(R, \Sigma)$. Assume that $X \to Y \in \Sigma^+$. Define R' as a relation schema such that $sort(R') = X \cup Y$, $\Sigma' = \Sigma_{XY}$, and let I' be $\pi_{XY}(I)$. Note that $I' \in inst(R', \Sigma')$. We use Lemma 3.5.2 to show the following.

Lemma 3.5.3 *Let $t_0 \in I$, $t'_0 = \pi_{XY}(t_0)$ and $A \in X \cup Y$. If $t_0[A]$ is the p-th element in I and $t'_0[A]$ is the p'-th element in I', then* $\text{INF}_I(p \mid \Sigma) \leq \text{INF}_{I'}(p' \mid \Sigma')$.

PROOF: Assume that $\|I\| = n$, $X \cup Y = \{A_1, \ldots, A_m\}$ and $\{t[X] \mid t \in I\}$ contains l tuples $\{\bar{c}_1, \ldots, \bar{c}_l\}$. For every $i \in [1, l]$, choose a tuple $t_i \in I$ such that $t_i[X] = \bar{c}_i$. Without loss of generality, assume that $t_0 = t_l$, $A = A_m$ and $t_i[A_j]$ is the $((i-1)m+j)$-th element in I. Thus, $t_1[A_1]$ is the first element in I, $t_1[A_m]$ is the m-th element in I and $t_l[A_m]$ is the lm-th element in I. We note that $p = lm$.

For every $\bar{a} = (a_1, \ldots, a_{p-1}, a_{p+1}, \ldots, a_n) \in \Omega(I, p)$, define $\bar{a}^* = (a_1, \ldots, a_{p-1}, v_{p+1}, \ldots, v_n)$, that is, $\bar{a}^*$ is generated from $\bar{a}$ by replacing each a_i ($i \in [p+1, n]$) by a variable. Furthermore, define $\Omega^*(I, p)$ as $\{\bar{a} \in \Omega(I, p) \mid$ for every $i \in [p+1, n]$, a_i is a variable$\}$. It is easy to see that if $\lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a|\bar{a})} = 1$, then $\lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}^*) \log \frac{1}{P(a|\bar{a}^*)} = 1$. Thus, by Lemma 3.5.2, for every $\bar{a} \in \Omega(I, p)$:

$$\lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \;\leq\; \lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}^*) \log \frac{1}{P(a \mid \bar{a}^*)}.$$

Therefore,

$$\begin{aligned}
\textsc{Inf}_I(p \mid \Sigma) &= \lim_{k\to\infty} \frac{1}{\log k} \sum_{\bar{a}\in\Omega(I,p)} \frac{1}{2^{n-1}} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \\
&= \frac{1}{2^{n-1}} \sum_{\bar{a}\in\Omega(I,p)} \lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \\
&\leq \frac{1}{2^{n-1}} 2^{n-p} \sum_{\bar{a}\in\Omega^*(I,p)} \lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \\
&= \frac{1}{2^{p-1}} \sum_{\bar{a}\in\Omega^*(I,p)} \lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})}. \qquad (3.8)
\end{aligned}$$

Observe that $\|I'\| = lm$. Without loss of generality assume that $p' = lm = p$. Then for every $\bar{a} = (a_1, \ldots, a_{p-1}, a_{p+1}, \ldots, a_n) \in \Omega(I, p)$, define $\bar{a}' \in \Omega(I', p')$ as $(a_1, \ldots, a_{p'-1})$. As in the case of $\bar{a}^*$, it is easy to see that $\lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a|\bar{a})} \leq \lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}') \log \frac{1}{P(a|\bar{a}')}$. Particularly, this property holds for every $\bar{a} \in \Omega^*(I, p)$. Thus, by (3.8) we conclude that

$$\begin{aligned}
\textsc{Inf}_{I'}(p' \mid \Sigma') &= \lim_{k\to\infty} \frac{1}{\log k} \sum_{\bar{a}\in\Omega(I',p')} \frac{1}{2^{p'-1}} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \\
&= \frac{1}{2^{p'-1}} \sum_{\bar{a}\in\Omega(I',p')} \lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \\
&\geq \frac{1}{2^{p-1}} \sum_{\bar{a}\in\Omega^*(I,p)} \lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \\
&\geq \textsc{Inf}_I(p \mid \Sigma).
\end{aligned}$$

□

PROOF OF THEOREM 3.5.1: First, we notice that adding new relations and constraints over them to a schema does not affect the information content of the old positions. Namely, let $S = \{R_1, \ldots, R_m\}$ be a relational schema, $\Sigma = \Sigma_1 \cup \cdots \cup \Sigma_m$ be a set of FDs over S such that Σ_i is a set of FDs over R_i ($i \in [1, m]$), $S' = \{R_1\}$, $\Sigma' = \Sigma_1$, $I \in \mathit{inst}(S, \Sigma)$ and $I' \in \mathit{inst}(S', \Sigma')$ such that $I' = I(R_1)$. Furthermore, let p be a position in $I(R_1)$ and p' the corresponding position in I'. Then $\textsc{Inf}_I(p \mid \Sigma) = \textsc{Inf}_{I'}(p' \mid \Sigma')$. The theorem now is a direct consequence of this fact and Lemma 3.5.3. □

A decomposition algorithm is *effective* in I if for one of its basic steps, and for some p, the inequality in Theorem 3.5.1 is strict: that is, the amount of information increases. This notion leads to another characterization of BCNF.

Proposition 3.5.4 (R, Σ) *is in BCNF if and only if no decomposition algorithm is effective in* (R, Σ).

Proof: ($\Rightarrow$) If (R, Σ) is in BCNF, then for every $I \in \mathit{inst}(R, \Sigma)$ and $p \in \mathit{Pos}(I)$, $\textsc{Inf}_I(p \mid \Sigma) = 1$. Thus, no decomposition algorithm can be effective on any $I \in \mathit{inst}(R, \Sigma)$.

($\Leftarrow$) Assume that (R, Σ) is not in BCNF. We will show that there exists a decomposition algorithm effective in (R, Σ).

Given that (R, Σ) is not in BCNF, we can find nonempty pairwise disjoint sets of attributes X, Y, Z such that $X \cup Y \cup Z = \mathit{sort}(R)$, $X \rightarrow Y \in \Sigma^+$, X is not a key and (XY, Σ_{XY}) is in BCNF. Let I be a database instance of R containing two tuples t_1, t_2 defined as follows. For every $A \in \mathit{sort}(R)$, $t_1[A] = 1$. If $X \rightarrow A \in \Sigma^+$, then $t_2[A] = 1$, otherwise $t_2[A] = 2$. It is easy to see that $I \in \mathit{inst}(R, \Sigma)$. Furthermore, for every $A \in Y$ and $p \in \mathit{Pos}(I)$ such that $t_1[A]$ (or $t_2[A]$) is the p-th element in I, $\textsc{Inf}_I(p \mid \Sigma) < 1$ and $\textsc{Inf}_{I_{XY}}(\pi_{XY}(p) \mid \Sigma_{XY}) = 1$ (since (XY, Σ_{XY}) is in BCNF). Therefore, $\textsc{Inf}_I(p \mid \Sigma) < \textsc{Inf}_{I_{XY}}(\pi_{XY}(p) \mid \Sigma_{XY})$. Thus, a decomposition algorithm that decomposes I into I_{XY} and I_{XZ} is effective in (R, Σ). □

Chapter 4

XML Databases

The goal of this dissertation is to find principles for good XML data design, and algorithms to produce such designs. To this end, in the previous chapter we have introduced an information-theoretic measure for testing when a relational normal form corresponds to a good design, and we have used this measure to provide information-theoretic justification for familiar normal forms such as BCNF and 4NF. Our intention is to extend this measure to XML databases and use it to provide justification for XML normal forms. In this chapter, we take a first step towards this goal by introducing a formal model for XML databases and a language for XML keys and foreign keys.

4.1 Introduction

XML (Extensible Markup Language) is a simple and flexible text format. It was originally designed for publishing electronic data, but today it has emerged as the standard language for storing and interchanging data on the web [ABS00].

An XML document is shown in Figure 4.1. This document contains two different types of tags: *start-tags*, such as `<course>` and `<title>`, and *end-tags*, such as `</course>` and `</title>`. These tags must be balanced and they are used to delimit *elements*. For example,

```
<title> Computer Organization </title>
```

is an element bounded by matching tags `<title>` and `</title>`. Every element can contain raw text, other elements, or a mixture of them. For instance, the element mentioned above contains raw text while the element delimited by `<ut>` con-

```
<ut>
  <student sno="st1">
    <name> John Smith </name>
    <taking>
      <course_number> CSC258 </course_number>
    </taking>
    <taking>
      <course_number> CSC309 </course_number>
    </taking>
  </student>
  <course cno="CSC258" dept="Computer Science">
    <title> Computer Organization </title>
    <enrolled>
      <student_number> st1 </student_number>
    </enrolled>
  </course>
</ut>
```

Figure 4.1: An XML document.

tains two elements, the first of which is delimited by tag `<student>`. In this case, `<student>` is a *sub-element* of `<ut>`. Elements can also contain *attributes*, such as `<course cno="CSC258" dept="Computer Science">`. This element contains two attributes: `cno` with value `CSC258` and `dept` with value `Computer Science`. The document shown in Figure 4.1 is part of a database for storing information about students and courses in the University of Toronto. Each `<student>` element represents a particular student, which has a name and a student number (`sno`) and is taking some courses. Each `<course>` element represents a particular course, which is given by some department (`dept`) and has a title, a course number (`cno`) and some students enrolled.

XML documents have a nested structure. This gives a lot of flexibility when storing information. For example, the name of a student in the XML document shown in Figure 4.1 is stored as an element containing raw text: `<name> John Smith </name>`. However, a nested structure can be used in these elements in order to distinguish first names from last names:

```
<name>
```

```
<!DOCTYPE ut [
  <!ELEMENT ut (student*, course*)>
  <!ELEMENT student (name, taking*)>
    <!ATTLIST student
              sno CDATA #REQUIRED>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT taking (course_number)>
  <!ELEMENT course_number (#PCDATA)>
  <!ELEMENT course (title, enrolled*)>
    <!ATTLIST course
              cno CDATA #REQUIRED
              dept CDATA #REQUIRED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT enrolled (student_number)>
  <!ELEMENT student_number (#PCDATA)>
]>
```

Figure 4.2: A DTD for a university database.

```
      <first> John </first>
      <last> Smith </last>
    </name>
```

To specify the structure of a class of XML documents, we have to specify, as in the case of relational databases, a *schema*. In the XML world, no standard way to do this has yet emerged, even though there are two predominant proposals: DTD (Document Type Definition) [Gol91, Hun00] and XML Schema [TBMM]. Even though DTDs are less expressive than XML Schema specifications, in general they are expressive enough for a large variety of applications [BNdB04]. Moreover, from a theoretical point of view, DTDs can be characterized in terms of unranked tree automata [Nev02], which have been widely studied in automata theory and more recently in database theory. In this dissertation, we consider only DTDs.

A DTD for the University of Toronto database is shown in Figure 4.2. This DTD specifies the elements allowed in XML documents by means of `ELEMENT` declarations. For example, `<student>` is an element since `<!ELEMENT student (name, taking*)>` appears in the DTD. An `ELEMENT` declaration also specifies the sub-elements of an element by means of a regular expression. For instance, `(name, taking*)` says that the sub-elements of `<student>` form a string in the regular language $name(taking)^*$ over the alphabet $\{name, taking\}$, that is, each `<student>` element has as sub-elements one `<name>` element followed by an arbitrary number of `<taking>` elements. `#PCDATA` is used to specify elements containing raw text, for instance `<!ELEMENT title (#PCDATA)>`, and an `ATTLIST` declaration is used to specify the attributes of an element. Finally, every document has an start-tag, which is called the *root* of the document and is specified by means of the `DOCTYPE` declaration (`<ut>` in the example).

In the next section, we formalize the notions of XML document and DTD.

4.2 XML Documents and DTDs

In this section, we present the formal model for XML documents and DTDs proposed by Fan and Libkin [FL01, FL02]. Also in this section, we introduce the notion of simple DTDs, which has been used to capture real-life DTDs [BNdB04], and we introduce the notion of path in XML documents and in DTDs.

Assume that we have the following disjoint sets: *El* of element names, *Att* of attribute names, *Str* of possible values of attributes and raw text, and *Vert* of node identifiers. All attribute names start with the symbol @, and these are the only ones starting with this symbol. We let S and $\perp$ (null) be reserved symbols not in any of those sets.

In Fan and Libkin's model [FL01, FL02], XML documents are represented as trees.

Definition 4.2.1 (XML Tree) *An* XML tree T *is defined to be a tree* $(V, lab, ele, att, root)$*, where*

- $V \subseteq Vert$ *is a finite set of* vertices *(*nodes*).*
- $lab : V \to El$*.*
- $ele : V \to Str \cup V^*$*.*
- att *is a partial function* $V \times Att \to Str$*. For each* $v \in V$*, the set* $\{@l \in Att \mid att(v, @l)$ *is defined*$\}$ *is required to be finite.*
- $root \in V$ *is called* the root of T*.*

The parent-child edge relation on V*,* $\{(v_1, v_2) \in V \times V \mid v_2$ *occurs in* $ele(v_1)\}$*, is required to form a rooted tree.*

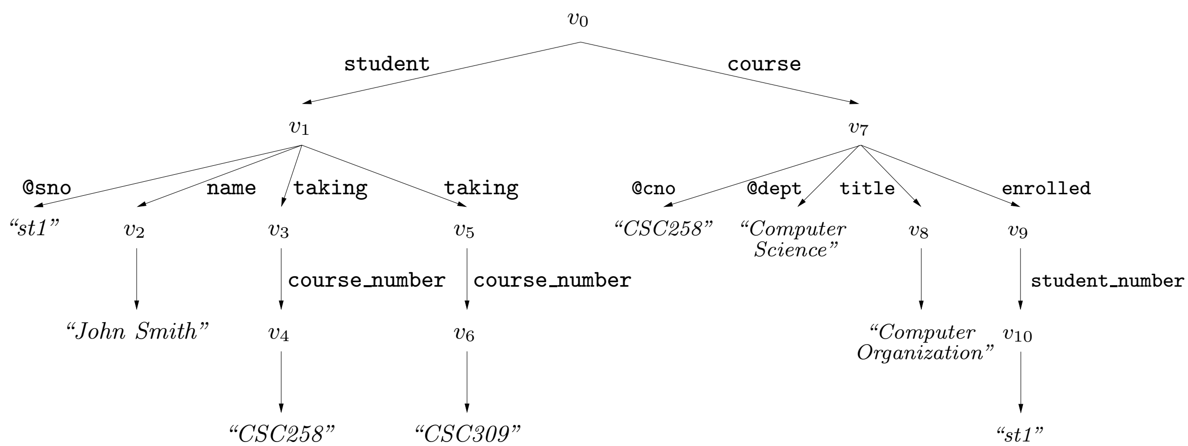


Figure 4.3: Tree representation of an XML document.

For every $x \in V$, $lab(x)$ is called the type of x in T. Notice that mixed content is not allowed in XML trees. The children of an element node can be either zero or more element nodes or one string.

In an XML tree T, for each $v \in V$, there is a unique path of parent-child edges from the root to v, and each node has at most one incoming edge. The root is a unique node. If a node x is labeled $\tau \in El$, then function *ele* defines the children of x and function *att* defines the attributes of x. The children of x are ordered. In contrast, its attributes are unordered and are identified by their labels (names).

In this dissertation, we also use the following notation. Given an XML tree T and an element type $\tau \in El$, $ext(\tau)$ is defined to be the set of all nodes of T of type τ. Furthermore, given a list of attributes $X = [@l_1, \ldots, @l_n]$, if v is a node of T such that $att(v, @l_i)$ is defined for every $i \in [1, n]$, then $v.@l_i$ is defined to be $att(v, @l_i)$ $(i \in [1, n])$ and $v[X]$ is defined to be the list of values $[att(v, @l_1), \ldots, att(v, @l_n)]$.

Example 4.2.2 Figure 4.3 shows the tree representation of the document shown in Figure 4.1. This tree contains a set of nodes $V = \{v_i \mid i \in [0, 10]\}$, which are labeled as follows.

$lab(v_0) = \texttt{ut}$	$lab(v_1) = \texttt{student}$	$lab(v_2) = \texttt{name}$
$lab(v_3) = \texttt{taking}$	$lab(v_4) = \texttt{course\_number}$	$lab(v_5) = \texttt{taking}$
$lab(v_6) = \texttt{course\_number}$	$lab(v_7) = \texttt{course}$	$lab(v_8) = \texttt{title}$
$lab(v_9) = \texttt{enrolled}$	$lab(v_{10}) = \texttt{student\_number}$	

Thus, $ext(\texttt{taking}) = \{v_3, v_5\}$ and $ext(\texttt{course\_number}) = \{v_4, v_6\}$. The structure of this tree is given by the function *ele*:

$ele(v_0) = [v_1, v_7]$ $\quad ele(v_1) = [v_2, v_3, v_5]$ $\quad ele(v_2) = [\mathit{John\ Smith}]$
$ele(v_3) = [v_4]$ $\quad ele(v_4) = [\mathit{CSC258}]$ $\quad ele(v_5) = [v_6]$
$ele(v_6) = [\mathit{CSC309}]$ $\quad ele(v_7) = [v_8, v_9]$ $\quad ele(v_8) = [\mathit{Computer\ Organization}]$
$ele(v_9) = [v_{10}]$ $\quad ele(v_{10}) = [\mathit{st1}]$

Moreover, this tree is rooted at v_0 ($root = v_0$) and it contains three attributes: $att(v_1, @sno) = \mathit{st1}$, $att(v_7, @cno) = \mathit{CSC258}$ and $att(v_7, @dept) = \mathit{Computer\ Science}$. Thus, $v_1.@sno = \mathit{st1}$, $v_7.@cno = \mathit{CSC258}$ and $v_7[@cno, @dept] = [\mathit{CSC258}, \mathit{Computer\ Science}]$. We note that the labels of the edges shown in Figure 4.3 are not part of the tree representation, they just represent the label of the vertices $\{v_1, \ldots, v_{10}\}$. $\square$

In Fan and Libkin's model [FL01, FL02], DTDs are defined as follows.

Definition 4.2.3 (DTD) *A* DTD *(Document Type Definition) is defined to be* $D = (E, A, P, R, r)$, *where:*

- $E \subseteq El$ *is a finite set of* element types.
- $A \subseteq Att$ *is a finite set of* attributes.
- P *is a mapping from* E *to* element type definitions*: Given* $\tau \in E$, $P(\tau) = \mathtt{S}$ *or* $P(\tau)$ *is a regular expression* α *defined as follows:*

$$\alpha \ ::= \ \epsilon \mid \tau' \mid \alpha|\alpha \mid \alpha, \alpha \mid \alpha^*$$

  where ϵ *is the empty sequence,* $\tau' \in E$, *and "|", "," and "∗" denote union, concatenation, and the Kleene closure, respectively.*

- R *is a mapping from* E *to the powerset of* A. *If* $@l \in R(\tau)$, *we say that* $@l$ *is* defined for τ.
- $r \in E$ *and is called* the element type of the root*. Without loss of generality, we assume that* $R(r) = \emptyset$ *and that* r *does not occur in* $P(\tau)$ *for any* $\tau \in E$.

The symbols ϵ and `S` represent element type declarations `EMPTY` and `#PCDATA`, respectively. In this dissertation, we also use the following shorthands for regular expressions: α^+ for (α, α^*) and $\alpha?$ for $(\epsilon|\alpha)$. We assume that each τ in $E - \{r\}$ is *connected to* r, i.e., either τ appears in $P(r)$, or it appears in $P(\tau')$ for some τ' that is connected to r.

Example 4.2.4 The DTD shown in Figure 4.2 is represented as follows. $E = \{ut$, $student$, $course$, $name$, $taking$, $course\_number$, $title$, $enrolled$, $student\_number\}$, $A = \{@sno$, $@cno$, $@dept\}$ and $r = ut$. Furthermore, $R(student) = \{@sno\}$, $R(course) = \{@cno$, $@dept\}$ and $R(\tau) = \emptyset$ for the remaining elements types τ, and P is defined as:

$$
\begin{array}{lcl@{\quad}lcl}
P(ut) & = & student^*, course^* & P(course) & = & title, enrolled^* \\
P(student) & = & name, taking^* & P(title) & = & \mathtt{S} \\
P(name) & = & \mathtt{S} & P(enrolled) & = & student\_number \\
P(taking) & = & course\_number & P(student\_number) & = & \mathtt{S} \\
P(course\_number) & = & \mathtt{S} & & &
\end{array}
$$

□

The notion of satisfaction of a DTD by an XML tree is defined in Fan and Libkin's formal model [FL01, FL02] as follows.

Definition 4.2.5 *Given a DTD $D = (E, A, P, R, r)$ and an XML tree $T = (V, lab, ele, att, root)$, we say that T* conforms to *D, denoted by $T \models D$, if*

- *lab is a mapping from V to E.*
- *For each $v \in V$, if $P(lab(v)) = \mathtt{S}$, then $ele(v) = [s]$, where $s \in Str$. Otherwise, $ele(v) = [v_1, \ldots, v_n]$, and the string $lab(v_1) \cdots lab(v_n)$ must be in the regular language defined by $P(lab(v))$.*
- *att is a partial function from $V \times A$ to Str such that for any $v \in V$ and $@l \in A$, $att(v, @l)$ is defined iff $@l \in R(lab(v))$.*
- *$lab(root) = r$.*

For example, the XML tree shown in Figure 4.3 conforms to the DTD shown in Figure 4.2.

A DTD D is called *recursive* if there is a cycle in the directed graph defined as $\{(\tau, \tau') \mid \tau'$ is in the alphabet of $P(\tau)\}$, and non-recursive otherwise. We also say that D is a *no-star* DTD if the Kleene star does not occur in any regular expression $P(\tau)$ (note that this is a stronger restriction than being $*$-free: a regular expression without the Kleene star yields a finite language, while the language of a $*$-free regular expression may still be infinite as it allows boolean operators including complement).

```
<!ELEMENT ProcessSpecification (Documentation*, SubstitutionSet*, (Include |
  BusinessDocument | ProcessSpecification | Package | BinaryCollaboration |
  BusinessTransaction | MultiPartyCollaboration)*)>
<!ELEMENT Include (Documentation*)>
<!ELEMENT BusinessDocument (ConditionExpression?, Documentation*)>
<!ELEMENT SubstitutionSet (DocumentSubstitution | AttributeSubstitution |
  Documentation)*>
<!ELEMENT BinaryCollaboration (Documentation*, InitiatingRole,
  RespondingRole, (Documentation | Start | Transition | Success | Failure |
  BusinessTransactionActivity | CollaborationActivity | Fork | Join)*)>
<!ELEMENT Transition (ConditionExpression?, Documentation*)>
```

Figure 4.4: Part of the Business Process Specification Schema of ebXML.

4.2.1 Simple DTDs

Typically, regular expressions used in DTDs are rather simple. We now formulate a criterion for simplicity that corresponds to a very common practice of writing regular expressions in DTDs [BNdB04, Cho02]. Given an alphabet E, a regular expression over E is called *trivial* if it is of the form $s_1, \ldots, s_n$, where for each s_i there is a letter $a_i \in E$ such that s_i is either a_i or $a_i?$ or a_i^+ or a_i^*, and for $i \neq j$, $a_i \neq a_j$. We call a regular expression s *simple* if there is a trivial regular expression s' such that any word w in the language denoted by s is a permutation of a word in the language denoted by s', and vice versa. Simple regular expressions were also considered in [ASV01] under the name of *multiplicity atoms*.

For example, $(a|b|c)^*$ is simple: a^*, b^*, c^* is trivial, and every word in $(a|b|c)^*$ is a permutation of a word in a^*, b^*, c^* and vice versa. Simple regular expressions are prevalent in DTDs [BNdB04]. For instance, every regular expression in the Business Process Specification Schema of ebXML [ebX], a set of specifications to conduct business over the Internet, is simple. Part of this schema is shown in Figure 4.4.

Definition 4.2.6 (Simple DTD) *A DTD D is* simple *if all productions in D use only simple regular expressions.*

In this dissertation, we turn our attention to simple DTDs when trying to either obtain more efficient algorithms for real-life DTDs or prove that a problem is infeasible even for real-life DTDs.

4.2.2 Paths in XML Documents and DTDs

Given an XML tree $T = (V, lab, ele, att, root)$ and a string $w = w_1 \cdots w_n$, with $w_1, \ldots, w_{n-1} \in El$ and $w_n \in El \cup Att \cup \{\mathtt{S}\}$, we say that w is a *path* in T if there are vertices $v_1, \ldots, v_{n-1}$ in V such that:

- $lab(v_i) = w_i$ $(1 \leq i \leq n-1)$,
- v_{i+1} is a child of v_i $(1 \leq i \leq n-2)$,
- if $w_n \in El$, then there is a child v_n of v_{n-1} such that $lab(v_n) = w_n$. If $w_n = @l$, with $@l \in Att$, then $att(v_{n-1}, @l)$ is defined. If $w_n = \mathtt{S}$, then v_{n-1} has a child in Str.

We assume that every path contains at least one element type. Thus, for example, *ut.student.name*, *ut.student.name.*S, *name.*S, *ut.course*, *course.@cno*, *course_number* and *ut.student.taking.course_number* are all paths[1] in the XML document shown in Figures 4.1 and 4.3.

We let $paths(T)$ stand for the set of paths in an XML tree T starting at the root of T. We note that for every node v of T, there exists a unique path in $paths(T)$ from the root of T to v. For example, from the set of paths shown above, only *ut.student.name*, *ut.student.name.*S, *ut.course* and *ut.student.taking.course_number* belong to $paths(T)$. Furthermore, given a pair of nodes x,y in T, with y a descendant of x, and a path $w = w_1 \cdots w_n$ in T, with w_n an element type, we say that w is a *path in T from x to y* if for the nodes $v_1, \ldots, v_n$ in the definition above we have that $v_1 = x$ and $v_n = y$.

Paths are an essential component of XML, as they have been used as one of the basic primitives in languages for navigating and querying XML documents [CD, BCF$^+$] and in data dependency languages for XML [BFW98, AV99, BDF$^+$01a, BDF$^+$01b]. In all these languages, the semantics of paths is as follows. Given an XML tree T, a node v of T, and a path w in T, $reach(v, w)$ is defined to be the set of all nodes and values in T reached by following w from v in this tree. Formally, if $w = w_1 \cdots w_n$ with $w_n \in El$, then $reach(v, w) = \{v' \mid w \text{ is a path in } T \text{ from } v \text{ to } v'\}$. Furthermore,

$$\begin{aligned} reach(v, w.@l) &= \bigcup_{v' \in reach(v,w)} \{att(v', @l)\}, \\ reach(v, w.\mathtt{S}) &= \bigcup_{v' \in reach(v,w)} \{s \in Str \mid ele(v') = [s]\}. \end{aligned}$$

[1] To improve the readability, we use the symbol . to separate the components of a path.

Thus, for example, in the XML tree shown in Figure 4.3 we have that:

$$\begin{aligned}
\mathit{reach}(v_0,\ \mathit{ut.student.name}) &= \{v_2\},\\
\mathit{reach}(v_0,\ \mathit{ut.student.name}.\mathtt{S}) &= \{\mathit{John\ Smith}\},\\
\mathit{reach}(v_0,\ \mathit{ut.student.taking.course_number}) &= \{v_4, v_6\},\\
\mathit{reach}(v_3,\ \mathit{course_number}) &= \{v_4\}.
\end{aligned}$$

Paths can also be defined over DTDs. More specifically, given a DTD $D = (E,\ A,\ P,\ R,\ r)$, a string $w = w_1 \cdots w_n$ is a *path* in D if w_i is in the alphabet of $P(w_{i-1})$, for each $i \in [2, n-1]$, and either w_n is in the alphabet of $P(w_{n-1})$ or $w_n = @l$ for some $@l \in R(w_{n-1})$. Furthermore, we say that $w_1 \cdots w_n$ is a path in D from τ to τ', where $\tau, \tau' \in E$, if $\tau = w_1$ and $\tau' = w_n$. We define $\mathit{length}(w)$ as n and $\mathit{last}(w)$ as w_n. We let $\mathit{paths}(D)$ stand for the set of all paths in D starting at the root, that is, the set of all paths $w = w_1 \cdots w_n$ such that $w_1 = r$, and we let $\mathit{EPaths}(D)$ stand for the set of all paths in $\mathit{paths}(D)$ that end with an element type (rather than an attribute or $\mathtt{S}$); that is, $\mathit{EPaths}(D) = \{w \in \mathit{paths}(D) \mid \mathit{last}(w) \in E\}$.

Finally, it is worth mentioning that a DTD D is recursive if and only if $\mathit{paths}(D)$ is infinite.

4.3 Keys and Foreign Keys for XML Databases

As in the case of relational databases, the design of XML databases is guided by the semantic information encoded in data dependencies. In this dissertation, we consider several flavors of the most popular XML data dependencies.

Although a number of dependency formalisms were developed for relational databases, functional and inclusion dependencies are the ones used most often. In fact, two subclasses of functional and inclusion dependencies, namely, keys and foreign keys, are most commonly found in practice. Both are fundamental to conceptual database design, and are supported by the SQL standard [MS93]. They provide a mechanism by which one can uniquely identify a tuple in a relation and refer to a tuple from another relation. They have proved useful in update anomaly prevention, query optimization and index design [AHV95, Ull88].

XML has become the prime standard for data exchange on the Web. XML data typically originates in databases. If XML is to represent data currently residing in databases, it should support keys and foreign keys, which are an essential part of the semantics of the data. Besides, keys and foreign keys for XML are important in, among other things, query optimization [PDST00], data integration [BGL+99, BM99, EM01b], and in data transformations between XML and relational databases [BCF+03, CFI+00, FK99, LC00, SSB+00, STZ+99, YP04].

A number of key and foreign key specifications have been proposed for XML. In Section 4.3.1, we introduce a key and foreign key language proposed by Fan and Siméon [FS00, FS03]. In Section 4.3.2, we extend this language to the case of relative constraints. Finally, in Section 4.3.3, we present other proposals for XML keys and foreign keys. It is worth mentioning that in Chapter 5, we extend the languages presented in this section to the case of constraints involving regular expressions, and in Chapter 6, we introduce a functional dependency language for XML.

4.3.1 Absolute keys and foreign keys

A class of absolute keys and foreign keys, denoted by $\mathcal{AC}^{*,*}_{K,FK}$ (we shall explain the notation shortly), was introduced by Fan and Siméon [FS00]. This class is defined for element types as follows. An $\mathcal{AC}^{*,*}_{K,FK}$-constraint φ over a DTD $D = (E, A, P, R, r)$ has one of the following forms:

- *Key*: $\tau[X] \to \tau$, where $\tau \in E$ and X is a nonempty set of attributes in $R(\tau)$. An XML tree T satisfies this constraint, denoted by $T \models \tau[X] \to \tau$, if T satisfies

$$\forall x, y \in ext(\tau)\ (x[X] = y[X] \to x = y).$$

- *Foreign key*: $\tau_1[X] \subseteq_{FK} \tau_2[Y]$, where $\tau_1, \tau_2 \in E$, X and Y are nonempty lists of attributes in $R(\tau_1)$ and $R(\tau_2)$, respectively, and $|X| = |Y|$. This constraint is satisfied by a tree T, denoted by $T \models \tau_1[X] \subseteq_{FK} \tau_2[Y]$, if $T \models \tau_2[Y] \to \tau_2$, and in addition T satisfies

$$\forall x \in ext(\tau_1)\ \exists y \in ext(\tau_2)\ (x[X] = y[Y]).$$

That is, $\tau[X] \to \tau$ says that the X-attribute values of a τ-element uniquely identify the element in $ext(\tau)$, and $\tau_1[X] \subseteq_{FK} \tau_2[Y]$ says that the Y-attribute values of a τ_2-element uniquely identify the element in $ext(\tau_2)$ and the list of X-attribute values of every τ_1-node in T must match the list of Y-attribute values of some τ_2-node in T. Notice that we use two notions of equality to define keys: value equality is assumed when comparing attributes, and node identity is used when comparing elements. We shall use the same symbol '=' for both, as it will never lead to ambiguity.

Example 4.3.1 Keys and foreign keys are defined in terms of XML attributes since `PCDATA` elements can always be replaced by attributes. For example, the `PCDATA` elements in the DTD shown in Figure 4.2 can be eliminated as follows:

```
<!DOCTYPE ut [
  <!ELEMENT ut (student*, course*)>
  <!ELEMENT student (taking*)>
    <!ATTLIST student
              sno CDATA #REQUIRED
              name CDATA #REQUIRED>
  <!ELEMENT taking (EMPTY)>
    <!ATTLIST taking
              course_number CDATA #REQUIRED>
  <!ELEMENT course (enrolled*)>
    <!ATTLIST course
              cno CDATA #REQUIRED
              dept CDATA #REQUIRED
              title CDATA #REQUIRED>
  <!ELEMENT enrolled (EMPTY)>
    <!ATTLIST enrolled
              student_number CDATA #REQUIRED>
]>
```

Subscripts K and FK in $\mathcal{AC}^{*,*}_{K,FK}$ denote keys and foreign keys, respectively, and the superscript '$*$' denotes multi-attribute. Constraints of $\mathcal{AC}^{*,*}_{K,FK}$ are generally referred to as *multi-attribute* constraints as they may be defined with multiple attributes. An $\mathcal{AC}^{*,*}_{K,FK}$ constraint is said to be *unary* if it is defined in terms of a single attribute; that is, $|X| = |Y| = 1$ in the above definition. In that case, we write $\tau.@l \to \tau$ for unary keys, and $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$ for unary foreign keys.

Example 4.3.2 To illustrate keys and foreign keys of $\mathcal{AC}^{*,*}_{K,FK}$, consider the DTD shown in Example 4.3.1. Typical $\mathcal{AC}^{*,*}_{K,FK}$-constraints over this DTD include:

$$\begin{aligned} \mathit{student}.@\mathit{sno} &\rightarrow \mathit{student}, \\ \mathit{course}.@\mathit{cno} &\rightarrow \mathit{course}, \\ \mathit{enroll}.@\mathit{student\_number} &\subseteq_{FK} \mathit{student}.@\mathit{sno}. \end{aligned}$$

The first two constraints are unary keys and the last constraint is a unary foreign key. The first constraint says that student number (*sno*) is an identifier for students, the second constraint says that course number (*cno*) is an identifier for courses, and the last constraint says that every person enrolled in a course must be a student.

We observe that if courses in different departments can have the same course number, then unary key $\mathit{course}.@\mathit{cno} \rightarrow \mathit{course}$ has to be replaced by a multi-attribute key:

$$\mathit{course}[@\mathit{cno}, @\mathit{dept}] \;\rightarrow\; \mathit{course}.$$

□

4.3.2 Relative keys and foreign keys

Since XML documents are hierarchically structured, one may be interested in the entire document as well as in its sub-documents. The latter give rise to *relative integrity constraints* [BDF+02, BDF+03], that only hold on certain sub-documents. Below we define relative keys and foreign keys. We use $\mathcal{RC}$ to denote such constraints. We use the notation $x \prec y$ when x and y are two nodes in an XML tree and y is a descendant of x.

A class of relative keys and foreign keys, denoted by $\mathcal{RC}^{*,*}_{K,FK}$, is defined as follows. An $\mathcal{RC}^{*,*}_{K,FK}$-constraint φ over a DTD $D = (E, A, P, R, r)$ has one of the following forms:

- *Relative key*: $\tau(\tau_1[X] \rightarrow \tau_1)$, where $\tau, \tau_1 \in E$ and X is a nonempty set of attributes in $R(\tau_1)$. It says that relative to each node x of element type τ, the set of attributes X is a key for all the τ_1-nodes that are descendants of x. That is, an XML tree T satisfies this constraint, denoted by $T \models \tau(\tau_1[X] \rightarrow \tau_1)$, if T satisfies

$$\forall x \in \mathit{ext}(\tau)\; \forall y, z \in \mathit{ext}(\tau_1)\; \big((x \prec y) \wedge (x \prec z) \wedge y[X] = z[X] \rightarrow y = z\big).$$

- *Relative foreign key*: $\tau(\tau_1[X] \subseteq_{FK} \tau_2[Y])$, where $\tau, \tau_1, \tau_2 \in E$, X and Y are nonempty lists of attributes in $R(\tau_1)$ and $R(\tau_2)$, respectively, and $|X| = |Y|$. It indicates that for each x in $\mathit{ext}(\tau)$, X is a foreign key of descendants of x of type

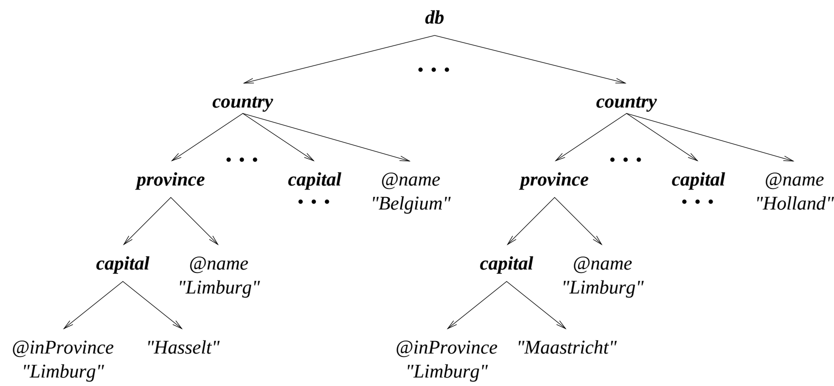


Figure 4.5: An XML document storing information about countries and their administrative subdivisions.

τ_1 that references a key Y of τ_2-descendants of x. That is, T satisfies φ, denoted by $T \models \tau(\tau_1[X] \subseteq_{FK} \tau_2[Y])$, if $T \models \tau(\tau_2[Y] \rightarrow \tau_2)$ and T satisfies

$$\forall x \in ext(\tau)\ \forall y \in ext(\tau_1)\ \big((x \prec y) \rightarrow \exists z \in ext(\tau_2)\ \ ((x \prec z) \wedge y[X] = z[Y])\big).$$

Note that relative constraints are somewhat related to the notion of keys for weak entities in relational databases (cf. [Ull88]). Also note that absolute constraints are a special case of relative constraints when $\tau = r$: i.e., $r(\tau[X] \rightarrow \tau)$ is the usual absolute key.

As in the case of absolute constraints, a relative constraint is said to be *unary* if it is defined in terms of a single attribute; that is, $|X| = |Y| = 1$ in the above definition. In that case, we write $\tau(\tau_1.@l \rightarrow \tau)$ for relative unary keys, and $\tau(\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2)$ for relative unary foreign keys.

Example 4.3.3 Consider an XML document that for each country lists its administrative subdivisions (e.g., into provinces or states), as well as capitals of provinces. A DTD is given below and an XML document conforming to it (represented as a tree) is depicted in Figure 4.5.

```
<!DOCTYPE db [
  <!ELEMENT db (country+)>
  <!ELEMENT country (province+, capital+)>
    <!ATTLIST country
              name CDATA #REQUIRED>
  <!ELEMENT province (capital, city*)>
```

```
    <!ATTLIST province
              name CDATA #REQUIRED>
  <!ELEMENT capital (#PCDATA)>
    <!ATTLIST capital
              inProvince CDATA #REQUIRED>
  <!ELEMENT city (#PCDATA)>
]>
```

Each country has a nonempty sequence of provinces and a nonempty sequence of province capitals, and for each province we specify its capital and perhaps other cities. Each country and province has an attribute *@name*, and each capital has an attribute *@inProvince*.

Now suppose we want to define keys for countries and provinces. One can state that country *@name* is a key for *country* elements. It is also tempting to say that *@name* is a key for *province*, but this may not be the case. The example in Figure 4.5 clearly shows that; which *Limburg* one is interested in probably depends on whether one's interests are in database theory, or in the history of the European Union. To overcome this problem, we define *@name* to be a key for *province relative* to a country; indeed, it is extremely unlikely that two provinces of the same country would have the same name. Thus, our constraints are:

$$country.@name \rightarrow country,$$
$$country(province.@name \rightarrow province),$$
$$country(capital.@inProvince \subseteq_{FK} province.@name).$$

The first constraint is like those we have encountered before: it is an *absolute* key, which applies to the entire document. The rest are *relative constraints* which are specified for sub-documents rooted at *country* elements. They assert that for each country, *@name* is a key of all *province* descendants of the country element and *@inProvince* is a foreign key referring to *@name* of *province* elements in the same sub-document. □

4.3.3 Related Work

We end this section by presenting other proposal of keys and foreign keys (inclusion dependencies) for XML. But before doing this, we need to introduce some terminology.

In all the proposals presented in this section, data dependencies for XML are defined as constraints on the values reached by following either paths or regular expressions in

XML trees. Recall that in Section 4.2.2 we define $reach(v, w)$ as the set of nodes of an XML tree T reached by following path w from node v. Here we extend this definition to the case of regular expressions. We define a *regular expression* over a finite alphabet Σ contained in $El \cup Att \cup \{\mathtt{S}\}$ as follows:

$$\beta \;\; ::= \;\; \epsilon \;\mid\; a \;\mid\; \beta.\beta \;\mid\; \beta \cup \beta \;\mid\; \beta^*,$$

where ϵ denotes the empty word, $a \in \Sigma$ and '.', '$\cup$' and '$*$' denote concatenation, union and Kleene closure, respectively. Then, given nodes x,y of an XML tree T, we say that y is reachable from x in T by following β if there is a string w in the regular language defined by β such that $y \in reach(x, w)$. The set of all such nodes is denoted by $reach(x, \beta)$. For example, in the XML document shown in Figures 4.1 and 4.3, $reach(v_0,\, ut.student.name.\mathtt{S})$ is the set of student names in the document, and

$$reach(v_0,\, ut.(student \cup taking \cup course)^*.(course\_number.\mathtt{S} \cup @cno))$$

is the set of all course numbers mentioned in the document.

One of the first kinds of data dependencies for XML was introduced by Abiteboul and Vianu [AV99]. They considered inclusion dependencies of the form $\beta \subseteq \gamma$, where β and γ are regular expressions. An XML tree T rooted at x satisfies this constraint if $reach(x, \beta) \subseteq reach(x, \gamma)$. For example, in the university database shown in Figure 4.1, the following constraint says that the set of courses taken by students is a subset of the set of courses given by the university:

$$ut.student.taken.course\_number.\mathtt{S} \;\; \subseteq \;\; ut.course.@cno.$$

An inclusion dependency $\beta \subseteq \gamma$ where β are γ are paths, like in the previous example, is called a *path constraint* [AV99]. Buneman et al. [BFW98] introduced a more expressive path constraint language. Given an XML tree and paths p_1, p_2, p_3 in T, in this language a constraint is an expression of either the *forward* form

$$\forall x \forall y\, (x \in reach(root, p_1) \wedge y \in reach(x, p_2) \rightarrow y \in reach(x, p_3)),$$

where *root* represents the root of the tree, or the *backward* form

$$\forall x \forall y\, (x \in reach(root, p_1) \wedge y \in reach(x, p_2) \rightarrow x \in reach(y, p_3)).$$

This language can be used to express relative constraints. For example, assume that the document shown in Figure 4.1 is extended to store information about students and courses in many different universities. Then `<ut>` is replaced by `<university>`:

```
<db>
  <university> ... </university>
  <university> ... </university>
  <university> ... </university>
</db>
```

Assume that we want to express the following constraint: for each university, the set of courses taken by its students is a subset of the set of courses given by that university. This dependency can be expressed as follows on the language of Buneman et al. [BFW98]:

$$\forall x \forall y\, (x \in \mathit{reach}(\mathit{root},\ \mathit{db.university}) \wedge \\ y \in \mathit{reach}(x,\ \mathit{student.taking.course\_number}.\mathtt{S}) \rightarrow y \in \mathit{reach}(x,\ \mathit{course}.@\mathit{cno})).$$

This constraint is relative to each university and, thus, it cannot be expressed by using Abiteboul and Vianu's path constraint language [AV99], which can only express constraints on the entire document (absolute constraints). By using Abiteboul and Vianu's approach, we can only say that if a student is taking a course, then this course is given in some university:

$$\mathit{db.university.student.taken.course\_number}.\mathtt{S} \quad \subseteq \quad \mathit{db.university.course}.@\mathit{cno}.$$

As we mention earlier, keys for XML were first considered by Fan and Siméon [FS00]. A key constraint language more expressive than Fan and Siméon's language was introduced by Buneman et al. [BDF$^+$01a, BDF$^+$01b]. This language allows the definition of absolute keys and relative keys. More precisely, an absolute key is an expression of the form $(\beta, \{\gamma_1, \ldots, \gamma_n\})$, where β, γ_1, ..., γ_n are regular expressions. If $n = 1$, then the key is said to be unary. An XML tree T satisfies this key if for every pair of node $x, y \in \mathit{reach}(\mathit{root}, \beta)$, if $\mathit{reach}(x, \gamma_i) \cap \mathit{reach}(y, \gamma_i) \neq \emptyset$, for every $i \in [1, n]$, then x and y are the same node. For example, a unary key dependency can be used to express that name is an identifier for students in the University of Toronto database: $(\mathit{ut.student}, \{\mathit{name}.\mathtt{S}\})$. But, if a nested structure is used in this database to distinguish first names from last names:

```
<ut>
  <student sno="st1">
    <name>
      <first> John </first>
```

```
      <last> Smith </last>
    </name>
    ...
  </student>
  ...
</ut>
```

then a non-unary key dependency is needed to characterize name as an identifier for students: $(ut.student, \{name.first.\mathtt{S},\ name.last.\mathtt{S}\})$.

Buneman et al. [BDF$^+$01a, BDF$^+$01b] defined a relative key as a pair of the form $(\beta_1, (\beta_2, \{\gamma_1, \ldots, \gamma_n\}))$, where β_1 is a regular expression and $(\beta_2, \{\gamma_1, \ldots, \gamma_n\})$ is an absolute key. An XML tree satisfies this key if every node reached from the root by following a path in β_1 satisfies $(\beta_2, \{\gamma_1, \ldots, \gamma_n\})$, that is, for every $x \in reach(root, \beta_1)$ and for every $y_1, y_2 \in reach(x, \beta_2)$, if $reach(y_1, \gamma_i) \cap reach(y_2, \gamma_i) \neq \emptyset$, for every $i \in [1, n]$, then y_1 and y_2 are the same node. For example, a relative key constraint can be used to express that a student cannot take the same course twice:

$$(ut.student,\ (taking,\ \{course\_number.\mathtt{S}\}).$$

This key dependency must be relative since two distinct students can take the same course.

Chapter 5

Consistency of XML Databases

The schema of an XML database consists of a type definition (a DTD) and a set of data dependencies. As opposed to relational databases, it has been shown previously that such schemas can be inconsistent in the sense that there is no way of populating the database and satisfying both the DTD and the set of data dependencies given by the schema. Inconsistent XML databases are poorly designed and, thus, it is desirable to have algorithms for checking consistency.

Since the goal of this dissertation is to find principles for good XML data design, and algorithms to produce such designs, in this chapter we study the consistency problem for XML databases. More specifically, we consider a variety of languages for XML keys and foreign keys, including the languages introduced in the previous chapter, and study the complexity of the consistency problem for these languages. Our main conclusion is that in the presence of foreign key constraints, compile-time verification of consistency is usually infeasible. We look at two types of constraints: absolute (that hold in the entire document), and relative (that only hold in a part of the document). For absolute constraints, we extend earlier decidability results to the case of primary multi-attribute keys and unary foreign keys, and to the case of constraints involving regular expressions, providing lower and upper bounds in both cases. For relative constraints, we show that even for unary constraints, the consistency problem is undecidable. At the end of the chapter, we use the results for both absolute and relative constraints to study the complexity of the consistency problems for real-life DTDs and XML Schema [TBMM].

It is worth mentioning that the consistency problem for functional dependencies is studied in Chapter 6.

5.1 Introduction

The schema of an XML database consists of a type definition (a DTD) and a set of data dependencies. A legitimate question then is whether such a specification is *consistent*, or meaningful: that is, whether there exists an XML document that both satisfies the constraints and conforms to the DTD.

In the relational database setting, such a question would have a trivial answer: one can write arbitrary (`primary`) `key` and `foreign key` specifications in SQL, without worrying about consistency. However, DTDs (and other schema specifications for XML) are more complex than relational schema and, consequently, DTDs may interact with keys and foreign keys in a rather nontrivial way, as shown in the following examples.

Example 5.1.1 As a simple example, consider the DTD given below:

```
<!DOCTYPE db [
  <!ELEMENT db (foo)>
  <!ELEMENT foo (foo)>
]>
```

Observe that there exists no finite XML tree conforming to this DTD, and hence this specification – that consists only of a DTD and no constraints – is inconsistent. □

Example 5.1.2 To illustrate the interaction between XML DTDs and key/foreign key constraints, consider a DTD D, which specifies a (nonempty) collection of teachers:

```
<!DOCTYPE teachers [
  <!ELEMENT teachers (teacher+)>
  <!ELEMENT teacher (teach, research)>
    <!ATTLIST teacher
              name CDATA #REQUIRED>
  <!ELEMENT teach (subject, subject)>
  <!ELEMENT research (#PCDATA)>
  <!ELEMENT subject (#PCDATA)>
    <!ATTLIST subject
              taught_by CDATA #REQUIRED>
]>
```

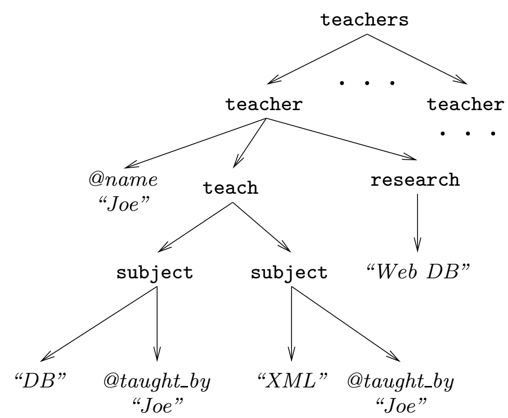


Figure 5.1: An XML tree for storing information about teachers.

It says that a teacher teaches two subjects and has an attribute *@name* and each subject has an attribute *@taught_by*. Consider a set Σ of key and foreign key constraints:

$$\begin{array}{rcl} teacher.@name & \to & teacher, \\ subject.@taught\_by & \to & subject, \\ subject.@taught\_by & \subseteq_{FK} & teacher.@name. \end{array}$$

Referring to an XML tree T, the first constraint asserts that two distinct *teacher* nodes in T cannot have the same *@name* attribute value: the (string) value of *@name* attribute uniquely identifies a *teacher* node. The second key states that the *@taught_by* attribute value uniquely identifies a *subject* node in T. The third constraint asserts that for every *subject* node x, there is a *teacher* node y in T such that the *@taught_by* attribute value of x equals the *@name* attribute value of y. Since *@name* is a key of *teacher*, the *@taught_by* attribute of any *subject* node refers to a unique *teacher* node.

Obviously, there exists an XML tree conforming to D, as shown in Figure 5.1. However, there is no XML tree that both conforms to D and satisfies Σ. To see this, recall that given an XML tree T and an element type τ, we use $ext(\tau)$ to denote the set of all the nodes labeled τ in T. Furthermore, given an attribute $@l$ of τ, assume that $values(\tau.@l)$ denotes the set of $@l$ attribute values of all τ elements. Then immediately from Σ follows a set of dependencies:

$$\begin{array}{rcl} |values(teacher.@name)| & = & |ext(teacher)|, \\ |values(subject.@taught\_by)| & = & |ext(subject)|, \\ |values(subject.@taught\_by)| & \leq & |values(teacher.@name)|, \end{array}$$

where $|\cdot|$ is the cardinality of a set. Therefore, we have

$$|ext(subject)| \leq |ext(teacher)|. \tag{5.1}$$

On the other hand, DTD D requires that each teacher must teach two subjects. Since no sharing of nodes is allowed in XML trees and the collection of *teacher* elements is nonempty, from D follows:

$$\begin{aligned} 1 &\leq |ext(teacher)|, && (5.2)\\ 2 \cdot |ext(teacher)| &= |ext(subject)|. && (5.3) \end{aligned}$$

Thus $|ext(teacher)| < |ext(subject)|$. Obviously, (5.1), (5.2) and (5.3) contradict each other and as an immediate result, there exists no XML document that both satisfies Σ and conforms to D. In particular, the XML tree in Figure 5.1 violates key constraint $subject.@taught\_by \rightarrow subject$. □

This example demonstrates that a DTD may impose dependencies on the cardinalities of certain sets of objects in XML trees. These *cardinality constraints* interact with keys and foreign keys. More specifically, keys and foreign keys also enforce cardinality constraints that interact with those imposed by a DTD. This makes the consistency analysis of keys and foreign keys for XML far more intriguing than its relational counterpart.

The constraints in this example are fairly simple: there is an immediate analogy between such XML constraints and relational keys and foreign keys. There have been a number of proposals for supporting more powerful keys and foreign keys for XML (e.g., [BDF$^+$02, TBMM]). Not surprisingly, the interaction between DTDs and those complicated XML constraints is more involved.

In light of this we are interested in the following family of the *consistency* (or *satisfiability*) problems, where $\mathcal{C}$ ranges over classes of integrity constraints:

PROBLEM	:	$\mathsf{SAT}(\mathcal{C})$.
INPUT	:	A DTD D, a set Σ of $\mathcal{C}$-constraints.
QUESTION	:	Is there an XML document that conforms to D and satisfies Σ?

In other words, we want to validate XML specifications statically, at compile-time. The main reason is twofold: first, complex interactions between DTDs and constraints are likely to result in inconsistent specifications, and second, an alternative dynamic approach to validation (simply check a document to see if it conforms to the DTD and satisfies the constraints) would not tell us whether repeated failures are due to a bad specification, or problems with the documents.

In this chapter we study the consistency problem for XML databases. More specifically, we consider a variety of languages for XML keys and foreign keys, including the languages introduced in the previous chapter, and study the complexity of the consistency problem for these languages. Our main conclusion is that in the presence of foreign key constraints, compile-time verification of consistency is usually infeasible. We look at two types of constraints: absolute (that hold in the entire document), and relative (that only hold in a part of the document). For absolute constraints, we extend earlier decidability results to the case of primary multi-attribute keys and unary foreign keys, and to the case of unary constraints involving regular expressions, providing lower and upper bounds in both cases. For relative constraints, we show that even for unary constraints, the consistency problem is undecidable. At the end of the chapter, we use the results for both absolute and relative constraints to study the complexity of the consistency problems for real-life DTDs and XML Schema [TBMM].

This chapter is organized as follows. In Section 5.2, we present the main results of Fan and Libkin [FL02] on the complexity of the consistency problem for absolute keys and foreign keys. In Section 5.3, we study the complexity of the consistency problem for the class of absolute multi-attribute keys and unary foreign keys, and the class of regular expression constraints which is an extension of absolute constraints with regular expressions. In Section 5.4, we investigate the complexity of the consistency problem for relative keys and foreign keys. In Section 5.5, we present two applications of the main results of this chapter. First, in Section 5.5.1, we study the complexity of the consistency problem for real-life DTDs. Then, in Section 5.5.2, we investigate the complexity of the consistency problem for XML Schema. Finally, in Section 5.6 we identify some directions for future research.

5.2 Known Results about the Consistency Problem

To the best of our knowledge, consistency of XML constraints in the presence of schema specifications was only investigated by Fan and Libkin [FL01, FL02]. In this section, we present their main results. More specifically, we point out the complexity of the consistency problem for the class $\mathcal{AC}^{*,*}_{K,FK}$ of keys and foreign keys introduced in Section 4.3, and we also point out the complexity of this problem for the following subclasses of $\mathcal{AC}^{*,*}_{K,FK}$: the class $\mathcal{AC}_{K,FK}$ consisting of unary keys and foreign keys, the class $\mathcal{AC}_{PK,FK}$ consisting of primary unary keys and foreign keys and the class $\mathcal{AC}^{*}_{K}$ consisting only of multi-attribute keys.

The following result shows that, in general, it is not possible to verify statically whether an XML specification is consistent.

Theorem 5.2.1 (Fan and Libkin [FL02]) $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ *is undecidable.*

This theorem was proved in [FL02] by showing that the implication problem associated with keys and foreign keys in relational databases is undecidable, and then reducing (the complement of) the implication problem to the consistency problem for $\mathcal{AC}^{*,*}_{K,FK}$ constraints.

Given this negative result, it is desirable to find some restrictions on $\mathcal{AC}^{*,*}_{K,FK}$ that lead to decidable cases. One important subclass of $\mathcal{AC}^{*,*}_{K,FK}$ is $\mathcal{AC}_{K,FK}$. A cursory examination of existing XML specifications reveals that most keys and foreign keys are single-attribute constraints, i.e., unary. The exact complexity of $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ was established in [FL02] by showing that this problem is polynomially equivalent to linear integer programming [Pap81]. Given that linear integer programming is known to be NP-complete [GJ79], the following theorem is an immediate consequence of the polynomial equivalence of the two problems.

Theorem 5.2.2 (Fan and Libkin [FL02]) $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ *is NP-complete.*

We have to be careful when interpreting this result, and, in general, when interpreting the complexity results presented in this dissertation. If we assume that PTIME $\neq$ NP, then from a theoretical point of view this result says that for every $\mathsf{SAT}(\mathcal{AC}_{K,FK})$-algorithm, there exists some XML specification (D, Σ) for which the algorithm is not going to be able to verify whether (D, Σ) is consistent in a reasonable amount of time. But given that XML specifications tend to be relatively small in practice, as opposed to XML documents which can be very large, and given that today we can find SAT solvers like BerkMin [GN02] and Chaff [MMZ$^+$01, ZM02] that routinely solve NP problems with thousands of variables, we can expect that in practice we are going to be able to verify whether XML specifications are consistent. Thus, given that all the NP-completeness results in this dissertation depend on the size of XML specifications, and not in the size of XML documents, we can expect them to be solvable in practice. Even more, for the case of PSPACE problems in this dissertation, we can also expect them to be solvable in some practical cases since today we can find model checkers that routinely solve PSPACE problems with hundreds of variables [VW94, DGV99, Hol03].

Since all the flavors of the consistency problem presented so far are intractable, we next want to find suitable restrictions that admit polynomial-time algorithms. A restriction found in many real-life examples is that of primary keys: for each element type, at most one key is defined. Unfortunately, as shown in [FL02], this restriction does not admit a polynomial-time algorithm.

Theorem 5.2.3 (Fan and Libkin [FL02]) $\mathsf{SAT}(\mathcal{AC}_{PK,FK})$ *is NP-complete.*

An interesting special case of low complexity involves keys only. It was shown in [FL02] that given a DTD D and a set Σ of multi-attributes keys over D, there exists an XML document conforming to D and satisfying Σ if and only if there exists an XML document conforming to D. Thus, the consistency problem for multi-attribute keys can be reduced to the consistency problem for DTDs, which in turn can be reduced in linear time to the emptiness problem for context free grammars. Since the latter problem can be solved in linear time (cf. [HU79]), the following theorem is obtained in [FL02].

Theorem 5.2.4 (Fan and Libkin [FL02]) $\mathsf{SAT}(\mathcal{AC}^{*}_{K})$ *is decidable in linear time.*

5.3 Absolute Integrity Constraints

In the previous section, we present the main results of [FL02]: the consistency problem for multi-attribute keys and foreign keys, $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$, is undecidable while the consistency problem for absolute unary keys and foreign keys, $\mathsf{SAT}(\mathcal{AC}_{K,FK})$, is NP-complete. These results only revealed the tip of the iceberg, as many other flavors of XML constraints exist, and are likely to be added to future standards for XML such as XML Schema [TBMM]. Our goals in this section is to study such constraints. In particular, in this section we establish the decidability and lower bounds for $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ and $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$, the consistency problems for primary multi-attribute keys and unary foreign keys and for regular unary keys and foreign keys. The class $\mathcal{AC}^{reg}_{K,FK}$ is an extension of $\mathcal{AC}_{K,FK}$ with regular expressions, which will be defined shortly.

5.3.1 Consistency of Multi-attribute Keys

We know that $\mathsf{SAT}(\mathcal{AC}_{K,FK})$, the consistency problem for absolute unary keys and foreign keys, is NP-complete [FL02]. In contrast, $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is undecidable [FL02]. This leaves a rather large gap: namely, $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$, where only keys are allowed to be multi-attribute (note that since a key is part of a foreign key, the other restriction, to $\mathcal{AC}^{1,*}_{K,FK}$, does not make sense).

The reason for the undecidability of $\mathsf{SAT}(\mathcal{AC}^{*,*}_{K,FK})$ is that the implication problem for functional and inclusion dependencies can be reduced to it [FL02]. However, this implication problem is known to be decidable – in fact, in cubic time – for single-attribute inclusion dependencies [CKV90], thus giving us hope to get decidability for multi-attribute keys and unary foreign keys. While the decidability of the consistency problem for $\mathcal{AC}^{*,1}_{K,FK}$ is still an open problem, we resolve a closely-related problem, $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$. That is, the consistency problem for *primary* multi-attribute keys and unary foreign keys. Recall that a set Σ of $\mathcal{AC}^{*,1}_{K,FK}$-constraints is said to be *primary* if for each element type τ, there is at most one key in Σ defined for τ-elements (including key dependencies defined by foreign key constraints). We prove the decidability by showing that complexity-wise, the problem is equivalent to a certain extension of integer linear programming studied in [GMWK02]:

PROBLEM:	PDE (Prequadratic Diophantine Equations)
INPUT:	An integer $n \times m$ matrix A, a vector $\vec{b} \in \mathbb{Z}^n$, and a set $E \subseteq \{1, \ldots, m\}^3$.
QUESTION:	Is there a vector $\vec{x} \in \mathbb{N}^m$ such that $A\vec{x} \leq \vec{b}$ and $x_i \leq x_j \cdot x_k$ for all $(i, j, k) \in E$.

Note that for $E = \emptyset$, this is exactly the integer linear programming problem [Pap81]. Thus, PDE can be thought of as integer linear programming extended with inequalities of the form $x \leq y \cdot z$ among variables. It is therefore NP-hard, and [GMWK02] proved an NEXPTIME upper bound for PDE. The exact complexity of the problem remains unknown.

Recall that two problems P_1 and P_2 are *polynomially equivalent* if there are PTIME reductions from P_1 to P_2 and from P_2 to P_1. We now show the following (the proof of the theorem is given in Appendix B.1).

Theorem 5.3.1 $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ *and PDE are polynomially equivalent.*

It is known that the linear integer programming problem is NP-hard [GJ79] and PDE is in NEXPTIME [GMWK02]. Thus from Theorem 5.3.1 follows immediately:

Corollary 5.3.2 $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ *is NP-hard, and can be solved in NEXPTIME.*

Obviously we cannot obtain the exact complexity of $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ without resolving the corresponding question for PDE, which appears to be quite hard [GMWK02]. The result of Theorem 5.3.1 can be generalized to *disjoint* $\mathcal{AC}^{*,1}_{K,FK}$-constraints: that is, a set Σ of $\mathcal{AC}^{*,1}_{K,FK}$-constraints in which for every element type τ and every two distinct keys $\tau[X] \to \tau$ and $\tau[Y] \to \tau$ in Σ (including key dependencies defined by foreign key constraints), $X \cap Y = \emptyset$. The proof of Theorem 5.3.1 applies almost verbatim to show the following.

Corollary 5.3.3 *The restriction of* $\mathsf{SAT}(\mathcal{AC}^{*,1}_{K,FK})$ *to disjoint constraints is polynomially equivalent to PDE and, thus, it is NP-hard and can be solved in NEXPTIME.*

5.3.2 Consistency of Regular Expression Constraints

Specifications of $\mathcal{AC}^{*,*}_{K,FK}$-constraints are associated with element types. To capture the hierarchical nature of XML data, constraints can also be defined on a collection of elements identified by a regular path expression. It is common to find path expressions in query languages for XML (e.g., XQuery [BCF+], XSL [Cla]). We define a *regular (path) expression* over a set of element types E as follows:

$$\beta \;\; ::= \;\; \epsilon \;\mid\; \tau \;\mid\; \beta.\beta \;\mid\; \beta \cup \beta \;\mid\; \beta^*,$$

where ϵ denotes the empty word, τ is an element type in E and '.', '$\cup$' and '$*$' denote concatenation, union and Kleene closure, respectively. A regular expression defines a language over the alphabet E, which will be denoted by β as well. Given a DTD $D = (E, A, P, R, r)$ and a regular expression β over E, we say that β is a *regular (path) expression over* D if β is of the form $r.\beta'$ where β' does not include r. In this section, we use '_' as a shorthand for $E - \{r\}$.

Recall that any pair of nodes x, y in an XML tree T with y a descendant of x uniquely determines the path, denoted by $\rho(x, y)$, from x to y. Also, recall that in Section 4.3.3, we say that y is reachable from x by following a regular expression β if and only if $\rho(x, y) \in \beta$. In that section, we denote by *reach*(x, β) the set of all nodes reachable from x by following β. For any fixed T, let *nodes*(β) stand for the set of nodes reachable from the root by following the regular expression β: $nodes(\beta) = reach(x, \beta)$, where x is the root of T. Note that for any element type $\tau \in E - \{r\}$, $nodes(r.\_^*.\tau) = ext(\tau)$.

We now define XML keys and foreign keys with regular expressions. Let DTD $D = (E, A, P, R, r)$.

- A *key* over D is an expression φ of the form $\beta.\tau.@l \to \beta.\tau$, where $\tau \in E, @l \in R(\tau)$, and β is a regular expression over D. An XML tree T satisfies φ, denoted by $T \models \varphi$, if for every $x, y \in nodes(\beta.\tau)$, $x.@l = y.@l$ implies $x = y$.

- A *foreign key* over D is an expression φ of the form $\beta_1.\tau_1.@l_1 \subseteq_{FK} \beta_2.\tau_2.@l_2$, where for $i = 1, 2$, $\tau_i \in E$, $@l_i \in R(\tau_i)$, and β_i is a regular expression over D. Here $T \models \varphi$ if $T \models \beta_2.\tau_2.@l_2 \to \beta_2.\tau_2$, and for every $x \in nodes(\beta_1.\tau_1)$ there exists $y \in nodes(\beta_2.\tau_2)$ such that $x.@l_1 = y.@l_2$.

We use $\mathcal{AC}^{reg}_{K,FK}$ to denote the set of all unary constraints defined with regular expressions. We do not consider multi-attribute constraints here, since they subsume $\mathcal{AC}^{*,*}_{K,FK}$ (by using $r.\_^*.\tau$ for τ), and thus consistency is undecidable for them.

Example 5.3.4 Consider an XML document in Figure 5.2, which conforms to the following DTD for schools:

```
<!ELEMENT r (students, courses, faculty, labs)>
<!ELEMENT students (student+)>
<!ELEMENT courses (cs340, cs108, cs434)>
<!ELEMENT faculty (prof+)>
<!ELEMENT labs (dbLab, pcLab)>
<!ELEMENT student (record)>                    /* similarly for prof
<!ELEMENT cs434 (takenBy+)>                    /* similarly for cs340, cs108
<!ELEMENT dbLab (acc+)>                        /* similarly for pcLab
```

Here we omit the descriptions of elements whose type is string (PCDATA). Assume that each *record* element has an attribute $@id$, each *takenBy* has an attribute $@sid$ (for student id), and each *acc* has an attribute $@num$. One may impose the following constraints over the DTD of that document:

$$\begin{aligned} r.\_^*.(student \cup prof).record.@id &\to r.\_^*.(student \cup prof).record, \\ r.\_^*.cs434.takenBy.@sid &\subseteq_{FK} r.\_^*.student.record.@id, \\ r.\_^*.dbLab.acc.@num &\subseteq_{FK} r.\_^*.cs434.takenBy.@sid. \end{aligned}$$

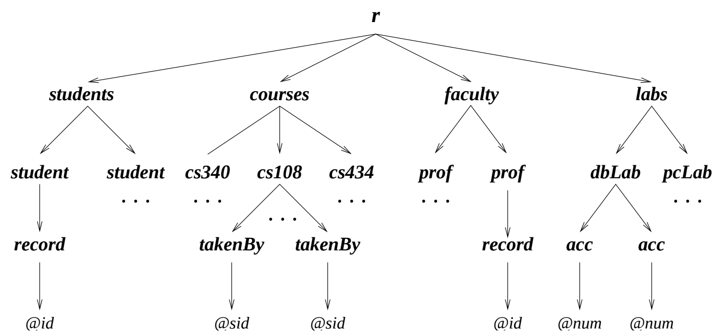


Figure 5.2: An XML document for storing information about students and professors.

Recall that '_' is a wildcard that matches any label except for r and '$\_^*$' is its Kleene closure that matches any path. The first constraint says that *@id* is a key for all records of students and professors. The other constraints specify foreign keys, which assert that cs434 can only be taken by students, and only students who are taking cs434 can have an account in the database lab. Recall that a foreign key also imposes a key constraint on the target elements, e.g., the last foreign key above also says that *@sid* is a key for students taking cs434.

Clearly, there is an XML tree satisfying both the DTD and the constraints. XML specifications are rarely written at once. Now suppose a new requirement is discovered: all faculty members must have a dbLab account. Consequently, one adds a new foreign key:

$$r.faculty.prof.record.@id \quad \subseteq_{FK} \quad r.\_^*.dbLab.acc.@num.$$

However, this addition makes the whole specification inconsistent. This is because previous constraints postulate that *dbLab* users are students taking cs434, and no professor can be a student since *@id* is a key for both students and professors, while the new foreign key insists upon professors also being *dbLab* users and the DTD enforces at least one professor to be present in the document. Thus no XML document both conforms to the DTD and satisfies all the constraints. □

For $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$, we are able to establish both an upper and a lower bound. The lower bound already indicates that the problem is perhaps infeasible in practice, even for non-recursive no-star DTDs. Finding the precise complexity of the problem remains open, and does not appear to be easy. In fact, even the current proof of the upper bound is quite involved, and relies on combining the techniques from [FL02] for coding

Class	$\mathcal{AC}^{*,*}_{K,FK}$ [FL02]	$\mathcal{AC}^{*,1}_{PK,FK}$	$\mathcal{AC}^{reg}_{K,FK}$	$\mathcal{AC}_{K,FK}$ [FL02]	$\mathcal{AC}^{*}_{K}$ [FL02]
	multi-attribute keys and foreign keys	primary multi-attribute keys, unary foreign keys	unary regular constraints (keys, foreign keys)	unary keys and foreign keys	multi-attributes keys
Upper bound	undecidable	NEXPTIME	2-NEXPTIME	NP	linear time
Lower bound	undecidable	NP	PSPACE	NP	linear time

Table 5.1: Complexity of the consistency problem for absolute constraints

DTDs and constraints with integer linear inequalities, and from [AV99] for reasoning about constraints given by regular expressions by using the product automaton for all the expressions involved in the constraints.

Theorem 5.3.5

a) $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ *can be solved in 2-NEXPTIME.*

b) $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ *is PSPACE-hard, even for non-recursive no-star DTDs.*

The proof of this theorem is given in Appendix B.2.

5.3.3 Summary

Table 5.1 shows a summary of the complexity results for the consistency problem for absolute keys and foreign keys (we also included the main results from [FL02] in this table).

5.4 Relative integrity constraints

Since XML documents are hierarchically structured, one may be interested in the entire document as well as in its sub-documents. The latter gives rise to relative integrity constraints [BDF$^+$02, BDF$^+$03], that only hold on certain sub-documents. In this section we study the complexity of the consistency problem for such constraints.

Recall that $\mathcal{RC}^{*,*}_{K,FK}$ is the class of relative keys and foreign keys (see Section 4.3.2). Following the notations for $\mathcal{AC}$, we use $\mathcal{RC}_{K,FK}$ to denote the class of all relative unary keys and unary foreign keys; $\mathcal{RC}_{PK,FK}$ means the primary key restriction. For example, the constraints given in Example 4.3.3 over the country/province/capital DTD are instances of $\mathcal{RC}_{K,FK}$.

Recall that $\mathsf{SAT}(\mathcal{AC}_{K,FK})$, the consistency problem for absolute unary constraints, is NP-complete. Thus, one would be tempted to think that $\mathsf{SAT}(\mathcal{RC}_{K,FK})$, the consistency problem for relative unary constraints, is decidable as well. We show, however, in Section 5.4.1, that this is not the case. As a consequence, we obtain that the consistency problem is also undecidable for any extension of $\mathcal{RC}_{K,FK}$, in particular, for extensions including multi-attribute constraints or regular expression constraints. In Section 5.4.2, we show that the consistency problem for relative multi-attribute keys, $\mathsf{SAT}(\mathcal{RC}^*_K)$, can be solved in linear time.

5.4.1 Undecidability of consistency

We now show that there is an enormous difference between unary absolute constraints, where $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ is decidable in NP, and unary relative constraints. We consider the consistency problem $\mathsf{SAT}(\mathcal{RC}_{K,FK})$. Clearly, the problem is r.e.; it turns out that one cannot lower this bound.

Theorem 5.4.1 $\mathsf{SAT}(\mathcal{RC}_{K,FK})$ *is undecidable.*

The proof of this theorem is given in Appendix B.3. In this proof, all relative keys are primary. Thus, we obtain:

Corollary 5.4.2 $\mathsf{SAT}(\mathcal{RC}_{PK,FK})$ *is undecidable.*

5.4.2 A linear time decidable case

Exactly as in the case of absolute keys, it can be shown that given a DTD D and a set Σ of relative multi-attributes keys over D, there exists an XML document conforming to D and satisfying Σ if and only if there exists an XML document conforming to D. Thus, the consistency problem for relative multi-attribute keys can be reduced to the consistency problem for DTDs, which in turn can be reduced in linear time to the emptiness problem for context free grammars [FL02]. Since the latter problem can be solved in linear time (cf. [HU79]), the following theorem is obtained.

Theorem 5.4.3 $\mathsf{SAT}(\mathcal{RC}^*_K)$ *can be solved in linear time.*

Class	$\mathcal{RC}^{*,*}_{K,FK}$ [FL02]	$\mathcal{RC}_{K,FK}$	$\mathcal{RC}_{PK,FK}$	$\mathcal{RC}^{*}_{K}$
	multi-attribute keys and foreign keys	unary keys and foreign keys	primary unary keys and foreign keys	multi-attribute keys
Upper bound	undecidable	undecidable	undecidable	linear time
Lower bound	undecidable	undecidable	undecidable	linear time

Table 5.2: Complexity of the consistency problem for relative constraints

5.4.3 Summary

Table 5.2 shows a summary of the complexity results for the consistency problem for relative keys and foreign keys (we also included the main results from [FL02] in this table).

5.5 Two Applications

In this section, we use the results of the previous sections to study the complexity of the consistency problems for real-life DTDs and XML Schema [TBMM].

5.5.1 Consistency of Real-Life DTDs

Since users tend to use very simple regular expressions in DTDs [BNdB04, Cho02], it is a natural question whether the consistency problem for real-life DTDs can be solved efficiently. In Section 4.2.1, we define simple regular expressions, which corresponds to a very common practice of writing regular expressions in DTDs [BNdB04], and then we define simple DTDs as those that only use simple regular expressions in their element rules. In this section, we study the complexity of the consistency problem for simple DTDs. More specifically, we show that compile-time verification of consistency is infeasible even for this class of DTDs

We note that the lower bounds shown in the previous sections do not directly carry over to the case of simple DTDs. However, a careful examination of the proofs of these lower bounds gives us the desired results. First, a slight modification of Fan and Libkin's proof [FL02] of the undecidability of the consistency problem for absolute keys and foreign keys shows that the same problem remains undecidable in the case of simple DTDs.

Class	$\mathcal{AC}^{*,*}_{K,FK}$	$\mathcal{AC}_{K,FK}$	$\mathcal{RC}_{K,FK}$
	absolute multi-attribute keys and foreign keys	absolute unary keys and foreign keys	relative unary keys and foreign keys
Upper bound	undecidable	NP	undecidable
Lower bound	undecidable	NP	undecidable

Table 5.3: Complexity of the consistency problem for simple DTDs.

Corollary 5.5.1 *The consistency problem for simple DTDs and $\mathcal{AC}^{*,*}_{K,FK}$-constraints is undecidable.*

Second, the same proof of Fan and Libkin of the NP-hardness of the consistency problem for absolute unary keys and foreign keys shows that the same problem remains NP-hard for the case of simple DTDs.

Corollary 5.5.2 *The consistency problem for simple DTDs and $\mathcal{AC}_{K,FK}$-constraints is NP-complete.*

Finally, a slight modification of the proof of Theorem 5.4.1 shows that the consistency problem for simple DTDs and relative unary keys and foreign keys is also undecidable.

Corollary 5.5.3 *The consistency problem for simple DTDs and $\mathcal{RC}_{K,FK}$-constraints is undecidable.*

Table 5.3 shows a summary of the complexity results for the consistency problem for simple DTDs.

5.5.2 Consistency of XML Schema Specifications

All the results shown so far are for DTDs and keys and foreign keys. These days, one of the prime standards for specifying XML data is *XML Schema* [TBMM]. XML Schema defines both a type system and a class of integrity constraints. It supports a variety of atomic types (e.g., string, integer, float, double, byte), complex type constructs (e.g., sequence, choice) and inheritance mechanisms (e.g., extension, restriction).

The central problem investigated in this section is the consistency problem for XML Schema. Our main conclusion is that the semantics of keys and foreign keys in XML Schema makes the consistency analysis rather intricate and intractable. Indeed, all the hardness and undecidability results of the previous sections carry over to specifications of XML Schema. However, using a new technique, we show that the most important tractable case under the standard key semantics, become intractable under the semantics of XML Schema.

Given a DTD D and a set Σ of keys and foreign keys under the XML Schema semantics, it is possible to use the narrowing technique employed in the proof of Theorem 5.3.5 to construct in polynomial time an XML Schema X such that, (D, Σ) is consistent if and only if X is consistent. Thus, to establish lower bounds for the complexity of the consistency problem for XML Schema, in this section we consider the following technical problem: Given a DTD D and a set Σ of key and foreign keys under the XML Schema semantics, is there an XML tree conforming to D and satisfying Σ? In the next subsection, we present the syntax and semantics of keys and foreign keys in XML Schema. Then, in the last subsection, we use the results of the previous sections –and a new technique– to establish some lower bounds for the complexity of the consistency problem for XML Schema.

Keys and Foreign Keys in XML Schema

Given a DTD $D = (E, A, P, R, r)$, a *key over* D is a constraint of the form:

$$P[Q_1, \ldots, Q_n] \quad \rightarrow \quad P,$$

where $n \geq 1$ and P, Q_1, …, Q_n are regular expressions over the alphabet $E \cup A \cup \{\mathtt{S}\}$. If $n = 1$, then the key is called unary. Expression P is called the *selector* of the key and is a regular expression conforming to the following BNF grammar [TBMM]:

$$\begin{array}{lcl} \mathit{selector} & ::= & \mathit{path} \;\mid\; \mathit{path} \cup \mathit{selector} \\ \mathit{path} & ::= & \mathit{root}//\mathit{sequence} \;\mid\; \mathit{sequence} \\ \mathit{sequence} & ::= & \tau \;\mid\; \_ \;\mid\; \mathit{sequence}/\mathit{sequence} \end{array}$$

Here _ is a wildcard that matches any element type, $\tau \in E$ and // represents the Kleene closure of _, that is, any possible finite sequence of node labels. The expressions Q_1, …, Q_n are called the *fields* of the key and are regular expressions conforming to the following BNF grammar [TBMM]:

$$\begin{array}{lcl} \mathit{field} & ::= & \mathit{path} \;\mid\; \mathit{path} \cup \mathit{field} \\ \mathit{path} & ::= & //\mathit{sequence}/\mathit{last} \;\mid\; /\mathit{sequence}/\mathit{last} \\ \mathit{sequence} & ::= & \epsilon \;\mid\; \tau \;\mid\; \_ \;\mid\; \mathit{sequence}/\mathit{sequence} \\ \mathit{last} & ::= & \mathtt{S} \;\mid\; @l \end{array}$$

Here @l is an attribute in A. This grammar differs from the one above in restricting the final step to match a text node or an attribute.

It should be mentioned that XML Schema expresses selectors and fields with *restricted* fragments of XPath [CD], which are precisely the regular expressions defined above. In XPath, '_' represents *child* and '//' denotes *descendant*[1].

A *foreign key* over a DTD D is an expression of the form:

$$P[Q_1, \ldots, Q_n] \quad \subseteq_{FK} \quad U[S_1, \ldots, S_n],$$

where P and U are the selectors of the foreign key, $n \geq 1$ and Q_1, ..., Q_n, S_1, ..., S_n are its fields. If $n = 1$, then the foreign key is called unary.

To define the semantics of keys and foreign keys in XML Schema, we need to introduce some terminology. In what follows we assume familiarity with the notation introduce in Section 5.3.2. Given an XML tree T conforming to a DTD D and a sequence of regular expressions P, Q_1, ..., Q_n over D such that P conforms to the BNF grammar for selectors and each Q_i ($i \in [1, n]$) conforms to the BNF grammar for fields and is of the form either $Q'_i/\mathtt{S}$ or $Q'_i/@l$, define the *qualified node set* of P, Q_1, ..., Q_n in T [TBMM], denoted by $\mathit{qns}(P, Q_1, \ldots, Q_n)$, as the set of nodes $x \in \mathit{nodes}(P)$ in T such that for every $i \in [1, n]$, there is exactly one node y_i such that $y_i \in \mathit{reach}(x, Q'_i)$ in T.

Now we are ready to define the semantics of keys and foreign keys in XML Schema. An XML tree T satisfies key dependency $P[Q_1, \ldots, Q_n] \rightarrow P$, denoted by $T \models P[Q_1, \ldots, Q_n] \rightarrow P$, if

1) $\mathit{nodes}(P) = \mathit{qns}(P, Q_1, \ldots, Q_n)$ in T.

2) For each $x_1, x_2 \in \mathit{nodes}(P)$ in T, if $\mathit{reach}(x_1, Q_i) = \mathit{reach}(x_2, Q_i)$ in T, for every $i \in [1, n]$, then $x_1 = x_2$.

That is, the values of Q_1, ..., Q_n uniquely identify the nodes reachable from the root by following path P. It further asserts that starting from each one of these nodes there is a single path conforming to the regular expression Q_i ($i \in [1, n]$).

An XML tree T satisfies a foreign key $P[Q_1, \ldots, Q_n] \subseteq_{FK} U[S_1, \ldots, S_n]$, denoted by $T \models P[Q_1, \ldots, Q_n] \subseteq_{FK} U[S_1, \ldots, S_n]$, if $T \models U[S_1, \ldots, S_n] \rightarrow U$ and

1) For each $x \in \mathit{qns}(P, Q_1, \ldots, Q_n)$ in T, there exists a node $y \in \mathit{nodes}(U)$ in T such that $\mathit{reach}(x, Q_i) = \mathit{reach}(y, S_i)$, for every $i \in [1, n]$.

[1] XPath [CD] uses '*' to denote wildcard. Here we use '_' instead to avoid overloading the symbol '*' with the Kleene star found in DTDs.

The foreign key asserts that $[S_1, \ldots, S_n]$ is a key for the nodes reachable by following path U and that for every node x reachable from the root by following path P such that $x \in qns(P, Q_1, \ldots, Q_n)$, there is a node y reachable from the root by following path U such that the Q_1, …, Q_n-values of x are equal to the S_1, …, S_n-values of y.

Checking Consistency of XML Schema Specifications

Now we are ready to show that the consistency check of XML Schema specifications is infeasible, even for specification containing only keys, the most important tractable case under the standard key semantics.

Observe that the definition of the semantics of keys and foreign keys in XML Schema requires the *uniqueness* and *existence* of the fields involved. Uniqueness conditions are required by the XML Schema semantics, but they are not present in various earlier proposals for XML keys coming from the database community [FS00, FL01, BDF$^+$02, BDF$^+$03]. Since these new conditions are trivially satisfied by the key and foreign key languages considered in Sections 5.2 and 5.3, we can use the results from these sections to prove lower bounds for the consistency problem for XML Schema. In particular, from Theorem 5.2.1 we obtain the undecidability of this problem.

Corollary 5.5.4 *The consistency problem for XML schema is undecidable.*

Furthermore, from Theorem 5.3.5 we obtain the intractability of the consistency problem for unary constraints.

Corollary 5.5.5 *The consistency problem for XML schema specifications containing only unary constraints is PSPACE-hard.*

Finally, using a new technique we show the intractability of the consistency problem for XML Schema specifications containing only keys.

Example 5.5.6 From Section 5.2, recall that given any DTD D and any set Σ of keys in $\mathcal{AC}^*_K$ over D, there exists an XML tree conforming to D and satisfying Σ if and only if there exists an XML tree conforming to D. Thus, any XML specification (D, Σ) where D is non-recursive and Σ is a set of keys in $\mathcal{AC}^*_K$ is consistent. We show here that a specification in XML Schema may not be consistent even for non-recursive DTDs in the absence of foreign keys.

Consider the following specification $S = (D, \Sigma)$ for biomedical data, where D is the following DTD:

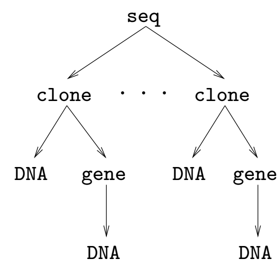


Figure 5.3: An XML document conforming to the DTD D shown in Example 5.5.6.

```
<!DOCTYPE seq [
  <!ELEMENT seq (clone+)>
  <!ELEMENT clone (DNA, gene)>
  <!ELEMENT gene (DNA)>
  <!ELEMENT DNA (#PCDATA)>
]>
```

and Σ contains only one key:

$$seq.clone.\_^*.DNA.\mathtt{S} \quad \rightarrow \quad seq.clone.$$

The DTD describes a nonempty sequence of *clone* elements: each *clone* has a *DNA* subelement and a *gene* subelement, and *gene* in turn has a *DNA* subelement, while *DNA* carries text data (PCDATA). The key in Σ attempts to enforce the following semantic information: there exist no two *clone* elements that have the same *DNA* no matter where the *DNA* appears as their descendant. We note that the syntax of XML Schema constraints is different from the syntax for XML constraints presented so far in that it allows a regular expression ($\_^*.DNA.\mathtt{S}$ in our example) to be the identifier of an element type.

This specification is inconsistent. XML Schema requires that for any XML document satisfying a key, the identifier (that is, $\_^*.DNA.\mathtt{S}$ in our example) must *exist* and be *unique*. However, as depicted in Figure 5.3, in any XML document that conforms to the DTD D, a *clone* element must have two *DNA* descendants. Thus, it violates the uniqueness requirement of the key in Σ. □

Fan and Libkin [FL02] showed that $\mathsf{SAT}(\mathcal{AC}^*_K)$, the consistency problem for absolute keys, is decidable in linear time. In Section 5.3.2, we introduce absolute unary keys and foreign keys with regular expressions. It is easy to extend this language to the case of

Class	multi-attribute constraints	unary constraints	unary keys
Lower bound	undecidable	PSPACE	NP

Table 5.4: Lower bounds for the complexity of the consistency problem for XML Schema.

multi-attribute constraints, and then it is easy to see that the same proof of Fan and Libkin can be used to show that the consistency problem for absolute keys involving regular expressions is also solvable in linear time. Even more, in Section 5.4.2 we use the same proof to show that $\mathsf{SAT}(\mathcal{RC}^*_K)$, the consistency problem for relative keys, is decidable in linear time. With all this evidence, one would be tempted to think that the consistency problem for keys under the XML Schema semantics can be solved efficiently. Somewhat surprisingly, this is not the case; the uniqueness and existence condition makes the problem intractable, even for unary keys.

Theorem 5.5.7 *The consistency problem for XML Schema specifications containing only unary keys is NP-hard.*

This result shows that the interaction of types and constraints under the XML Schema semantics is so intricate that the consistency check of XML Schema specifications is infeasible.

Table 5.4 shows a summary of the lower bounds for the consistency problem for XML Schema specifications.

5.6 Conclusions

We studied the problem of statically checking XML specifications, which may include various schema definitions as well as integrity constraints. As observed earlier, such static validation is quite desirable as an alternative to dynamic checking, which would attempt to validate each document; indeed, in the case of repeated failures, one does not know whether the problems lies in the documents or in the specification. Our main conclusion is that, however desirable, the static checking is hard: even with very simple document definitions given by DTDs, and (foreign) keys as constraints, the complexity ranges from NP-hard to undecidable.

Although most of the results of the chapter are negative, the techniques developed in the chapter help study consistency of individual XML specification with type and constraints. These techniques include, e.g., the connection between regular expression constraints and integer linear programming and automata.

One open problem is to close the complexity gaps. However, these are by no means trivial: for example, $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ was proved to be equivalent to a problem related to Diophantine equations whose exact complexity remains unknown. In the case of $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$, we think that it is more likely that our lower bound corresponds to the exact complexity of this problem. However, the algorithm is quite involved, and we do not yet see a way to simplify it to prove the matching upper bound.

Chapter 6

Functional Dependencies for XML

Up to this point, we have given a set of information-theoretic tools for testing when a condition on a relational database design, specified by a normal form, corresponds to a good design, we have used this measure to provide information-theoretic justification for familiar relational normal forms such as BCNF and 4NF, we have introduced a formal model for XML databases and we have investigated the consistency problem for XML schemas given by DTDs together with keys and foreign keys. Since the goal of this dissertation is to find principles for good XML data design, it is time to start studying the elements that we need to introduce a normal form for XML documents. More specifically, it is time to introduce functional dependencies (FDs) for XML documents, which are the basic component of the XML normal form proposed in this dissertation.

In this chapter, we introduce FDs for XML by considering a relational representation of documents and defining FDs on them. The relational representation is somewhat similar to the total unnesting of a nested relation (see Section 2.2); however, we have to deal with DTDs that may contain arbitrary regular expressions, and be recursive. Our representation via *tree tuples* may contain null values and, thus, XML FDs are introduced via FDs on incomplete relations [AM84, IJ84, LL98]. In this chapter, we also investigate the consistency and implication problems for XML functional dependencies.

6.1 Tree Tuples

To extend the notions of functional dependencies to the XML setting, we represent XML trees as sets of tuples. While various mappings from XML to the relational model have been proposed [FK99, STZ$^+$99], the mapping that we use is of a different nature, as our goal is not to find a way of storing documents efficiently, but rather find a correspondence between documents and relations that lends itself to a natural definition of functional dependency.

Various languages proposed for expressing XML integrity constraints such as keys, [BDF+01a, BDF+01b, TBMM], treat XML trees as unordered (for the purpose of defining the semantics of constraints): that is, the order of children of any given node is irrelevant as far as satisfaction of constraints is concerned. In XML trees, on the other hand, children of each node are ordered. Since the notion of functional dependency we propose also does not use the ordering in the tree, we first define a notion of subsumption that disregard this ordering.

Given two XML trees $T_1 = (V_1, lab_1, ele_1, att_1, root_1)$ and $T_2 = (V_2, lab_2, ele_2, att_2, root_2)$, we say that T_1 is subsumed by T_2, written as $T_1 \preceq T_2$ if

- $V_1 \subseteq V_2$.
- $root_1 = root_2$.
- $lab_2 \restriction_{V_1} = lab_1$.
- $att_2 \restriction_{V_1 \times Att} = att_1$.
- For all $v \in V_1$, $ele_1(v)$ is a sublist of a permutation of $ele_2(v)$.

This relation is a pre-order, which gives rise to an equivalence relation: $T_1 \equiv T_2$ iff $T_1 \preceq T_2$ and $T_2 \preceq T_1$. That is, $T_1 \equiv T_2$ iff T_1 and T_2 are equal as unordered trees. We define $[T]$ to be the $\equiv$-equivalence class of T. We write $[T] \models D$ if $T_1 \models D$ for some $T_1 \in [T]$. It is easy to see that for any $T_1 \equiv T_2$, $paths(T_1) = paths(T_2)$. We shall also write $T_1 \prec T_2$ when $T_1 \preceq T_2$ and $T_2 \not\preceq T_1$.

In Chapter 4 we gave the standard definition of a tree conforming to a DTD ($T \models D$). Here we also need a weaker version of T being compatible with D ($T \lhd D$).

Definition 6.1.1 *Given a DTD D and an XML tree T, we say that T is* compatible *with D (written $T \lhd D$) iff $paths(T) \subseteq paths(D)$.*

Clearly, $T \models D$ implies T is compatible with D. Furthermore, for any $T_1 \equiv T_2$, we have that $T_1 \lhd D$ iff $T_2 \lhd D$.

In the following definition we extend the notion of tuple for relational databases to the case of XML. In a relational database, a tuple is a function that assigns to each attribute a value from the corresponding domain. In our setting, a tree tuple t in a DTD D is a function that assigns to each path in D a value in $Vert \cup Str \cup \{\perp\}$ in such a way that t represents a finite tree with paths from D containing at most one occurrence of each path. In this section, we show that an XML tree can be represented as a set of tree tuples.

Definition 6.1.2 (Tree tuples) *Given a DTD* $D = (E, A, P, R, r)$, *a* tree tuple t in D *is a function from* $paths(D)$ *to* $Vert \cup Str \cup \{\perp\}$ *such that:*

- *For* $p \in EPaths(D)$, $t(p) \in Vert \cup \{\perp\}$, *and* $t(r) \neq \perp$.
- *For* $p \in paths(D) - EPaths(D)$, $t(p) \in Str \cup \{\perp\}$.
- *If* $t(p_1) = t(p_2)$ *and* $t(p_1) \in Vert$, *then* $p_1 = p_2$.
- *If* $t(p_1) = \perp$ *and* p_1 *is a prefix of* p_2, *then* $t(p_2) = \perp$.
- $\{p \in paths(D) \mid t(p) \neq \perp\}$ *is finite.*

$\mathcal{T}(D)$ *is defined to be the set of all tree tuples in* D. *For a tree tuple* t *and a path* p, *we write* $t.p$ *for* $t(p)$.

Example 6.1.3 Suppose that D is the DTD shown in Example 7.1.1 from Chapter 7. Then a tree tuple in D assigns values to each path in $paths(D)$:

$$
\begin{aligned}
&t(courses) = v_0\\
&t(courses.course) = v_1\\
&t(courses.course.@cno) = \texttt{csc200}\\
&t(courses.course.title) = v_2\\
&t(courses.course.title.\mathtt{S}) = \texttt{Automata Theory}\\
&t(courses.course.taken\_by) = v_3\\
&t(courses.course.taken\_by.student) = v_4\\
&t(courses.course.taken\_by.student.@sno) = \texttt{st1}\\
&t(courses.course.taken\_by.student.name) = v_5\\
&t(courses.course.taken\_by.student.name.\mathtt{S}) = \texttt{Deere}\\
&t(courses.course.taken\_by.student.grade) = v_6\\
&t(courses.course.taken\_by.student.grade.\mathtt{S}) = \texttt{A+}
\end{aligned}
$$

□

We intend to consider tree tuples in XML trees conforming to a DTD. The ability to map a path to null ($\perp$) allows one in principle to consider tuples with paths that do not reach the leaves of a given tree, although our intention is to consider only paths that do reach the leaves. However, nulls are still needed in tree tuples because of the disjunction in DTDs. For example, let $D = (E, A, P, R, r)$, where $E = \{r, a, b\}$, $A = \emptyset$, $P(r) = (a|b)$, $P(a) = \epsilon$ and $P(b) = \epsilon$. Then $paths(D) = \{r, r.a, r.b\}$ but no tree tuple coming from an XML tree conforming to D can assign non-null values to both $r.a$ and $r.b$.

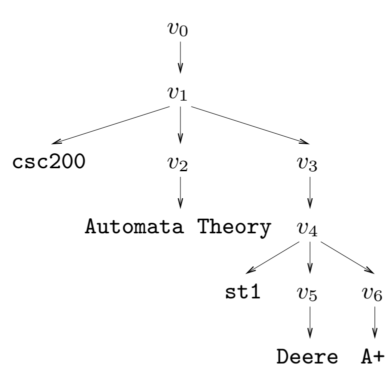
If D is a recursive DTD, then $paths(D)$ is infinite; however, only a finite number of values in a tree tuple are different from $\perp$. For each tree tuple t, its non-null values give rise to an XML tree as follows.

Definition 6.1.4 **($tree_D$)** *Given a DTD $D = (E, A, P, R, r)$ and a tree tuple $t \in \mathcal{T}(D)$, $tree_D(t)$ is defined to be an XML tree $(V, lab, ele, att, root)$, where $root = t.r$ and*

- $V = \{v \in Vert \mid \exists p \in paths(D)$ *such that* $v = t.p\}$.
- *If* $v = t.p$ *and* $v \in V$, *then* $lab(v) = last(p)$.
- *If* $v = t.p$ *and* $v \in V$, *then* $ele(v)$ *is defined to be the list containing* $\{t.p' \mid t.p' \neq \perp$ *and* $p' = p.\tau, \tau \in E$, *or* $p' = p.\mathtt{S}\}$, *ordered lexicographically.*
- *If* $v = t.p$, $@l \in A$ *and* $t.p.@l \neq \perp$, *then* $att(v, @l) = t.p.@l$.

We note that in this definition the lexicographic order is arbitrary, and it is chosen simply because an XML tree must be ordered.

Example 6.1.5 Let D be the DTD from Example 7.1.1 and t the tree tuple from Example 6.1.3. Then, t gives rise to the following XML tree:



□

Notice that the tree in the example conforms to the DTD from Example 7.1.1. In general, this need not be the case. For instance, if the rule `<!ELEMENT taken_by (student*)>` in the DTD shown in Example 7.1.1 is changed by a rule saying that every course must have at least two students `<!ELEMENT taken_by (student, student+)>`, then the tree shown in Example 6.1.5 does not conform to the DTD. However, $tree_D(t)$ would always be compatible with D, as easily follows from the definition:

Proposition 6.1.6 *If $t \in \mathcal{T}(D)$, then $tree_D(t) \lhd D$.*

We would like to describe XML trees in terms of the tuples they contain. For this, we need to select tuples containing the maximal amount of information. This is done via the usual notion of ordering on tuples (and relations) with nulls, [BJO91, Gra91, Gun92]. If we have two tree tuples t_1, t_2, we write $t_1 \sqsubseteq t_2$ if whenever $t_1.p$ is defined, then so is $t_2.p$, and $t_1.p \neq \perp$ implies $t_1.p = t_2.p$. As usual, $t_1 \sqsubset t_2$ means $t_1 \sqsubseteq t_2$ and $t_1 \neq t_2$. Given two sets of tree tuples, X and Y, we write $X \sqsubseteq^\flat Y$ if $\forall t_1 \in X \exists t_2 \in Y \; t_1 \sqsubseteq t_2$.

Definition 6.1.7 **(*$tuples_D$*)** *Given a DTD D and an XML tree T such that $T \lhd D$, $tuples_D(T)$ is defined to be the set of maximal, with respect to $\sqsubseteq$, tree tuples t such that $tree_D(t)$ is subsumed by T; that is:*

$$\max_{\sqsubseteq}\{t \in \mathcal{T}(D) \mid tree_D(t) \preceq T\}.$$

Observe that $T_1 \equiv T_2$ implies $tuples_D(T_1) = tuples_D(T_2)$. Hence, $tuples_D$ applies to equivalence classes: $tuples_D([T]) = tuples_D(T)$. The following proposition lists some simple properties of $tuples_D(\cdot)$.

Proposition 6.1.8 *If $T \lhd D$, then $tuples_D(T)$ is a finite subset of $\mathcal{T}(D)$. Furthermore, $tuples_D(\cdot)$ is monotone: $T_1 \preceq T_2$ implies $tuples_D(T_1) \sqsubseteq^\flat tuples_D(T_2)$.*

Proof: We prove only monotonicity. Suppose that $T_1 \preceq T_2$ and $t_1 \in tuples_D(T_1)$. We have to prove that there exists $t_2 \in tuples_D(T_2)$ such that $t_1 \sqsubseteq t_2$. If $t_1 \in tuples_D(T_2)$, this is obvious, so assume that $t_1 \notin tuples_D(T_2)$. Given that $t_1 \in tuples_D(T_1)$, $tree_D(t_1) \preceq T_1$, and, therefore, $tree_D(t_1) \preceq T_2$. Hence, by definition of $tuples_D(\cdot)$, there exists $t_2 \in tuples_D(T_2)$ such that $t_1 \sqsubset t_2$, since $t_1 \notin tuples_D(T_2)$. $\square$

Example 6.1.9 In Example 7.1.1 we saw a DTD D and a tree T conforming to D. In Example 6.1.3 we saw one tree tuple t for that tree, with identifiers assigned to some of the element nodes of T. If we assign identifiers to the rest of the nodes, we can compute the set $\mathit{tuples}_D(T)$ (the attributes are sorted as in Example 6.1.3):

$$\begin{aligned}\{\ &(v_0,\ v_1,\ \texttt{csc200}, v_2,\ \texttt{Automata Theory},\ v_3,\ v_4,\ \texttt{st1},\ v_5,\ \texttt{Deere},\ v_6,\ \texttt{A+}),\\ &(v_0,\ v_1,\ \texttt{csc200}, v_2,\ \texttt{Automata Theory},\ v_3,\ v_7,\ \texttt{st2},\ v_8,\ \texttt{Smith},\ v_9,\ \texttt{B-}),\\ &(v_0,\ v_{10},\ \texttt{mat100},\ v_{11},\ \texttt{Calculus I},\ v_{12},\ v_{13},\ \texttt{st1},\ v_{14},\ \texttt{Deere},\ v_{15},\ \texttt{A}),\\ &(v_0,\ v_{10},\ \texttt{mat100},\ v_{11},\ \texttt{Calculus I},\ v_{12},\ v_{16},\ \texttt{st3},\ v_{17},\ \texttt{Smith},\ v_{18},\ \texttt{B+})\ \}\end{aligned}$$

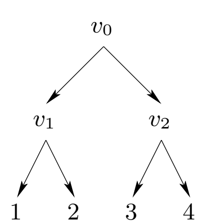
□

The following example shows that there is a direct correspondence between tuples in relational databases and tree tuples in XML documents.

Example 6.1.10 Assume that we are given a relation schema $S(A, B)$ and a simple DTD $D = (E, A, P, R, r)$, where $E = \{r, s\}$, $A = \{@a, @b\}$, $P(r) = s^*$, $P(s) = \epsilon$, $R(r) = \emptyset$ and $R(s) = \{@a, @b\}$. We can use XML trees conforming to D to code instances of S. For example, the following instance I:

A	B
1	2
3	4

can be coded as an XML tree $T = (V,\ \mathit{lab},\ \mathit{ele},\ \mathit{att},\ \mathit{root})$ conforming to D:



where $\mathit{lab}(v_0) = r$, $\mathit{lab}(v_1) = \mathit{lab}(v_2) = S$, $\mathit{att}(v_1, @a) = 1$, $\mathit{att}(v_1, @b) = 2$, $\mathit{att}(v_2, @a) = 3$ and $\mathit{att}(v_2, @b) = 4$.

The set of paths in D is $\{r, r.s, r.s.@a, r.s.@b\}$ and the set of tree tuples in T is:

r	$r.s$	$r.s.@a$	$r.s.@b$
v_0	v_1	1	2
v_0	v_2	3	4

Thus, there exists a one-to-one correspondence between the tuples in I and the tree tuples in T. $\square$

Finally, we define the trees represented by a set of tuples X as the minimal, with respect to $\preceq$, trees containing all tuples in X.

Definition 6.1.11 ($trees_D$) *Given a DTD D and a set of tree tuples $X \subseteq \mathcal{T}(D)$, $trees_D(X)$ is defined to be:*

$$\min_{\preceq}\{T \mid T \lhd D \textit{ and } \forall t \in X,\ \ tree_D(t) \preceq T\}.$$

Notice that if $T \in trees_D(X)$ and $T' \equiv T$, then T' is in $trees_D(X)$. The following shows that every XML document can be represented as a set of tree tuples, if we consider it as an unordered tree. That is, a tree T can be reconstructed from $tuples_D(T)$, up to equivalence $\equiv$.

Theorem 6.1.12 *Given a DTD D and an XML tree T, if $T \lhd D$, then $trees_D(tuples_D([T])) = [T]$.*

PROOF: Every XML tree is finite, and, therefore, $tuples_D([T]) = \{t_1, \ldots, t_n\}$, for some n. Suppose that $T \notin trees_D(\{t_1, \ldots, t_n\})$. Given that $tree_D(t_i) \preceq T$, for each $i \in [1, n]$, there is an XML tree T' such that $T' \preceq T$ and $tree_D(t_i) \preceq T'$, for each $i \in [1, n]$. If $T' \prec T$, there is at least one node, string or attribute value contained in T which is not contained in T'. This value must be contained in some tree tuple t_j ($j \in [1, n]$), which contradicts $tree_D(t_j) \preceq T'$. Therefore, $T \in trees_D(tuples_D([T]))$.

Let $T' \in trees_D(tuples_D([T]))$. For each $i \in [1, n]$, $tree_D(t_i) \preceq T'$. Thus, given that $tuples_D(T) = \{t_1, \ldots, t_n\}$, we conclude that $T \preceq T'$, and, therefore, by definition of $trees_D$, $T' \equiv T$. $\square$

Example 6.1.13 It could be the case that for some set of tree tuples X there is no an XML tree T such that for every $t \in X$, $tree(t) \preceq T$. For example, let D be a DTD $D = (E,\ A,\ P,\ R,\ r)$, where $E = \{r, a, b\}$, $A = \emptyset$, $P(r) = (a|b)$, $P(a) = \epsilon$ and $P(b) = \epsilon$. Let $t_1, t_2 \in \mathcal{T}(D)$ be defined as

$$\begin{array}{lclclcl} t_1.r & = & v_0 & \qquad & t_2.r & = & v_2 \\ t_1.r.a & = & v_1 & & t_2.r.a & = & \perp \\ t_1.r.b & = & \perp & & t_2.r.b & = & v_3 \end{array}$$

Since $t_1.r \neq t_2.r$, there is no an XML tree T such that $tree_D(t_1) \preceq T$ and $tree_D(t_2) \preceq T$.
$\square$

We say that $X \subseteq \mathcal{T}(D)$ is *D-compatible* if there is an XML tree T such that $T \lhd D$ and $X \subseteq tuples_D(T)$. For a D-compatible set of tree tuples X there is always an XML tree T such that for every $t \in X$, $tree_D(t) \preceq T$. Moreover,

Proposition 6.1.14 *If $X \subseteq \mathcal{T}(D)$ is D-compatible, then (a) There is an XML tree T such that $T \lhd D$ and $trees_D(X) = [T]$, and (b) $X \sqsubseteq^\flat tuples_D(trees_D(X))$.*

PROOF: (a) Assume that $D = (E, A, P, R, r)$. Since X is D-compatible, there exists an XML tree $T' = (V', lab', ele', att', root')$ such that $T' \lhd D$ and $X \subseteq tuples_D(T')$. We use T' to define an XML tree $T = (V, lab, ele, att, root)$ such that $trees_D(X) = [T]$.

For each $v \in V'$, if there is $t \in X$ and $p \in paths(D)$ such that $t.p = v$, then v is included in V. Furthermore, for each $v \in V$, $lab(v)$ is defined as $lab'(v)$, $ele(v) = [s_1, \ldots, s_n]$, where each $s_i = t'.p.\mathtt{S}$ or $s_i = t'.p.\tau$ for some $t' \in X$ and $\tau \in E$ such that $t'.p = v$. For each $@l \in A$ such that $t'.p.@l \neq \perp$ and $t'.p = v$ for some $t' \in X$, $att(v, @l)$ is defined as $t'.p.@l$. Finally, $root$ is defined as $root'$. It is easy to see that $trees_D(X) = [T]$.
(b) Let $t \in X$ and T be an XML tree such that $trees_D(X) = [T]$. If $t \in tuples_D([T])$, then the property holds trivially. Suppose that $t \notin tuples_D([T])$. Then, given that $tree_D(t) \preceq T$, there is $t' \in tuples_D([T])$ such that $t \sqsubset t'$. In either case, we conclude that there is $t' \in tuples_D(trees_D(X))$ such that $t \sqsubseteq t'$. $\square$

The example below shows that it could be the case that $tuples_D(trees_D(X))$ properly dominates X, that is, $X \sqsubseteq^\flat tuples_D(trees_D(X))$ and $tuples_D(trees_D(X)) \not\sqsubseteq^\flat X$. In particular, this example shows that the inverse of Theorem 6.1.12 does not hold, that is, $tuples_D(trees_D(X))$ is not necessarily equal to X for every set of tree tuples X, even if this set is D-compatible. Let D be as in Example 6.1.13 and $t_1, t_2 \in \mathcal{T}(D)$ be defined as

$$\begin{array}{lcllcl} t_1.r & = & v_0 & \qquad t_2.r & = & v_0 \\ t_1.r.a & = & v_1 & \qquad t_2.r.a & = & \perp \\ t_1.r.b & = & \perp & \qquad t_2.r.b & = & v_2 \end{array}$$

Let t_3 be a tree tuple defined as $t_3.r = v_0$, $t_3.r.a = v_1$ and $t_3.r.b = v_2$. Then, $tuples_D(trees_D(\{t_1, t_2\})) = \{t_3\}$ since $t_1 \sqsubset t_3$ and $t_2 \sqsubset t_3$, and, therefore, $\{t_1, t_2\} \sqsubseteq^\flat tuples_D(trees_D(\{t_1, t_2\}))$ and $tuples_D(trees_D(\{t_1, t_2\})) \not\sqsubseteq^\flat \{t_1, t_2\}$.

From Theorem 6.1.12 and Proposition 6.1.14, it is straightforward to prove the following Corollary.

Corollary 6.1.15 *For a D-compatible set of tree tuples X, $\mathit{trees}_D(\mathit{tuples}_D(\mathit{trees}_D(X)))$ $= \mathit{trees}_D(X)$.*

Theorem 6.1.12 and Proposition 6.1.14 are summarized in the diagram presented in the following figure. In this diagram, X is a D-compatible set of tree tuples. The arrow $\hookrightarrow$ stands for the $\sqsubseteq^\flat$ ordering.

$$\begin{tikzcd} X \arrow[r, hook, "\mathit{trees}_D"] \arrow[dr, hook] & {[T]} \arrow[d, shift right, "\mathit{tuples}_D"'] \\ & X' \arrow[u, shift right, "\mathit{trees}_D"'] \end{tikzcd}$$

6.2 Functional Dependencies

We define functional dependencies for XML by using tree tuples. For a DTD D, a *functional dependency (FD)* over D is an expression of the form $S_1 \to S_2$ where S_1, S_2 are finite non-empty subsets of $\mathit{paths}(D)$. The set of all FDs over D is denoted by $\mathcal{FD}(D)$.

For $S \subseteq \mathit{paths}(D)$, and $t, t' \in \mathcal{T}(D)$, $t.S = t'.S$ means $t.p = t'.p$ for all $p \in S$. Furthermore, $t.S \neq \perp$ means $t.p \neq \perp$ for all $p \in S$. If $S_1 \to S_2 \in \mathcal{FD}(D)$ and T is an XML tree such that $T \lhd D$ and $S_1 \cup S_2 \subseteq \mathit{paths}(T)$, we say that T *satisfies* $S_1 \to S_2$ (written $T \models S_1 \to S_2$) if for every $t_1, t_2 \in \mathit{tuples}_D(T)$, $t_1.S_1 = t_2.S_1$ and $t_1.S_1 \neq \perp$ imply $t_1.S_2 = t_2.S_2$. We observe that if tree tuples t_1, t_2 satisfy an FD $S_1 \to S_2$, then for every path $p \in S_2$, $t_1.p$ and $t_2.p$ are either both null or both non-null. Moreover, if for every pair of tree tuples t_1, t_2 in an XML tree T, $t_1.S_1 = t_2.S_1$ implies they have a null value on some $p \in S_1$, then the FD is trivially satisfied by T.

The previous definition extends to equivalence classes, since for any FD φ, and $T \equiv T'$, $T \models \varphi$ iff $T' \models \varphi$. We write $T \models \Sigma$, for $\Sigma \subseteq \mathcal{FD}(D)$, if $T \models \varphi$ for each $\varphi \in \Sigma$, and we write $T \models (D, \Sigma)$, if $T \models D$ and $T \models \Sigma$.

Example 6.2.1 Referring back to Example 7.1.1, we have the following FDs. `cno` is a key of `course`:

$$\mathit{courses.course.@cno} \to \mathit{courses.course}.$$

Another FD says that two distinct `student` subelements of the same `course` cannot have the same `sno`:

$$\{courses.course,\ courses.course.taken\_by.student.@sno\} \rightarrow \\ courses.course.taken\_by.student.$$

Finally, to say that two `student` elements with the same `sno` value must have the same `name`, we use

$$courses.course.taken\_by.student.@sno \rightarrow courses.course.taken\_by.student.name.\mathtt{S}.$$

□

We offer a few remarks on our definition of FDs. First, using the tree tuples representation, it is easy to combine node and value equality: the former corresponds to equality between vertices and the latter to equality between strings. Moreover, keys naturally appear as a subclass of FDs, and relative constraints can also be encoded. Note that by defining the semantics of $\mathcal{FD}(D)$ on $\mathcal{T}(D)$, we essentially define satisfaction of FDs on relations with null values, and our semantics is the standard semantics used in [AM84, LL98].

Given a DTD D and a set $\Sigma \cup \{\varphi\}$ of FDs over D, we say that φ is *implied* by (D, Σ), denoted by $(D, \Sigma) \vdash \varphi$, if for every XML tree T conforming to D and satisfying Σ, it is the case that $T \models \varphi$. The set of all FDs implied by (D, Σ) will be denoted by $(D, \Sigma)^+$. Furthermore, an FD φ is *trivial* if $(D, \emptyset) \vdash \varphi$. In relational databases, the only trivial FDs are $X \rightarrow Y$, with $Y \subseteq X$. Here, DTD forces some more interesting trivial FDs. For instance, for each $p \in \mathit{EPaths}(D)$ and p' a prefix of p, $(D, \emptyset) \vdash p \rightarrow p'$, and for every p, $p.@l \in \mathit{paths}(D)$, $(D, \emptyset) \vdash p \rightarrow p.@l$. As a matter of fact, trivial functional dependencies in XML documents can be much more complicated than in the relational case, as we show in the following example.

Example 6.2.2 Let $D = (E, A, P, R, r)$ be a DTD. Assume that a, b and c are element types in D and $P(r) = (a|b|c)$. Then, for every $p \in \mathit{paths}(D)$, $\{r.a, r.b\} \rightarrow p$ is a trivial FD since for every XML tree T conforming to D and every tree tuple t in T, $t.r.a = \perp$ or $t.r.b = \perp$. □

6.3 The Implication Problem for XML Functional Dependencies

In the next Chapter, we introduce a normal form for XML specifications given by DTDs and functional dependencies. As in the case of relational databases, testing whether an XML specification is in this normal form involves testing some conditions on the functional dependencies implied by the constraints in the specification. In this section, we study the implication problem for XML functional dependencies.

Although XML FDs and relational FDs are defined similarly, the implication problem for the former class is far more intricate. In Section 6.3.1, we show that the implication problem for XML functional dependencies is decidable in co-NEXPTIME. Then we present classes of DTDs for which this problem can be solved more efficiently. In Section 6.3.2, we show that the implication problem for simple DTDs (see Section 4.2) can be solved in quadratic time. In Section 6.3.3, we introduce a class of DTDs that contains the class of simple DTDs and for which the implication problem can still be solved efficiently. These classes include most of real-world DTDs. In Section 6.3.4 we introduce two classes of DTDs for which the implication problem is coNP-complete. Finally, in Section 6.3.5 we show that, unlike relational FDs, XML FDs are not finitely axiomatizable. In all these sections we assume, without loss of generality, that all FDs have a single path on the right-hand side.

6.3.1 The General Case

In this section, we establish the decidability of the implication problem for XML functional dependencies and DTDs.

Theorem 6.3.1 *The implication problem for XML functional dependencies over DTDs is solvable in co-NEXPTIME.*

Proof: See Appendix C.1. □

6.3.2 Simple regular expressions

In this section, we show that the implication problem for simple DTDs (see Section 4.2) can be solved in quadratic time.

Theorem 6.3.2 *The implication problem for FDs over simple DTDs is solvable in quadratic time.*

PROOF SKETCH: Here we present the sketch of the proof. The complete proof can be found in Appendix C.2.

In the first part of the proof we show that given a simple DTD D and a set of FDs $\Sigma \cup \{S \to p\}$ over D, the problem of verifying whether $\Sigma \not\vdash S \to p$ can be reduced to the problem of finding a counterexample to a certain implication problem. That is, we need to find an XML tree T such that $T \models (D, \Sigma)$, $T \not\models S \to p$, T contains two tree tuples and T satisfies some additional conditions that depend on the simplicity of D. Essentially, if an element type is allowed to occur zero times ($a?$ or a^*), then in constructing the counterexample such elements not need to be considered if they are irrelevant to the functional dependencies under consideration. Furthermore, all the element types in a regular expression in D can be considered independently. Observe that this condition is not longer valid if a regular expression in D contains a disjunction (D is not simple). For instance, if $(a|b)$ is a regular expression in D, then a and b are not independent; if a does not appear in an XML tree conforming to D, then b appears in this tree.

In the second part of the proof we show that the problem of finding this counterexample can be reduced to the problem of verifying if a certain propositional formula φ, constructed from D and $\Sigma \cup \{S \to p\}$, is satisfiable. This formula is of the form $\varphi_1 \vee \cdots \vee \varphi_n$, where n is at most the length of the path p and each φ_i ($i \in [1, n]$) is a conjunction of Horn clauses and is of linear size in the size of D and $\Sigma \cup \{S \to p\}$. Given that the consistency problem for Horn clauses is solvable in linear time [DG84], we conclude that the counterexample can be found in quadratic time and, therefore, our original problem can be solved in quadratic time. □

6.3.3 Small number of disjunctions

In a simple DTD, disjunction can appear in expressions of the form $(a|\epsilon)$ or $(a|b)^*$, but a general disjunction $(a|b)$ is not allowed. For example, the following DTD cannot be represented as a simple DTD:

```
<!DOCTYPE university [
  <!ELEMENT university (course*)>
  <!ELEMENT course (number, student*)>
```

```
  <!ELEMENT number (#PCDATA)>
  <!ELEMENT student ((name | FLname), grade)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT FLname (first_name, last_name)>
  <!ELEMENT first_name (#PCDATA)>
  <!ELEMENT last_name (#PCDATA)>
  <!ELEMENT grade (#PCDATA)>
]>
```

In this example, every student must have a name. This name can be an string or it can be a composition of a first and a last name. It is desirable to express constraints on this kind of DTDs. For instance,

$$\mathit{student}.\mathit{name}.\mathtt{S} \rightarrow \mathit{student},$$

$$\{\mathit{student}.\mathit{FLname}.\mathit{first\_name}.\mathtt{S}, \mathit{student}.\mathit{FLname}.\mathit{last\_name}.\mathtt{S}\} \rightarrow \mathit{student},$$

are functional dependencies in this domain. It is also desirable to reason about these constraints efficiently. Often, a DTD is not simple because a small number of regular expressions in it are not simple. In this section we will show that there is a polynomial time algorithm for reasoning about constraints over DTDs containing a small number of disjunctions.

A regular expression s over an alphabet A is a *simple disjunction* if $s = \epsilon$, $s = a$, where $a \in A$, or $s = s_1|s_2$, where s_1, s_2 are simple disjunctions over alphabets A_1, A_2 and $A_1 \cap A_2 = \emptyset$. A DTD $D = (E, A, P, R, r)$ is called *disjunctive* if for every $\tau \in E$, $P(\tau) = s_1, \ldots, s_m$, where each s_i is either a simple regular expression or a simple disjunction over an alphabet A_i ($i \in [1, m]$), and $A_i \cap A_j = \emptyset$ ($i, j \in [1, m]$ and $i \neq j$). This generalizes the concept of a simple DTD.

With each disjunctive DTD D, we associate a number N_D that measures the complexity of unrestricted disjunctions in D. Formally, for a simple regular expression s, $N_s = 1$. If s is a simple disjunction, then N_s is the number of symbols $|$ in s plus 1. If $P(\tau) = s_1, \ldots, s_n$, then N_τ is 1, if $s_1, \ldots, s_n$ is a simple regular expression, $N_\tau = |\{p \in \mathit{paths}(D) \mid \mathit{last}(p) = \tau\}| \times N_{s_1} \times \cdots \times N_{s_n}$ otherwise. Finally, $N_D = \prod_{\tau \in E} N_\tau$.

Theorem 6.3.3 *For any fixed $c > 0$, the FD implication problem for disjunctive DTDs D with $N_D \leq \|D\|^c$ is solvable in polynomial time*[1].

[1] $\|\cdot\|$ is the size of the description of an object. For instance, $\|p\|$ is the length of the path p and $\|S\|$ is the sum of the lengths of the paths in S.

Proof sketch: Here we present the sketch of the proof. The complete proof can be found in Appendix C.3.

The main idea of this proof is that the implication problem for disjunctive DTDs can be reduced to a number of implication problems for simple DTDs by splitting the disjunctions. More precisely, given a disjunctive DTD D and a set of functional dependencies $\Sigma \cup \{S \to p\}$ over D, there exist (D_1, Σ_1), ..., (D_n, Σ_n) such that each D_i ($i \in [1, n]$) is a simple DTD, Σ_i is a set of functional dependencies over D_i ($i \in [1, n]$) and $(D, \Sigma) \vdash S \to p$ if and only if $(D_i, \Sigma_i) \vdash S \to p$ for every $i \in [1, n]$. The number n of implication problems for simple DTDs is at most N_D. Thus, since the implication problem for simple DTDs can be solved in quadratic time (see Theorem 6.3.2), the implication problem for disjunctive DTDs D with $N_D \leq \|D\|^c$, for some constant c, can be solved in polynomial time. $\square$

6.3.4 Relational DTDs

There are some classes of DTDs for which the implication problem is not tractable. One such class consists of arbitrary disjunctive DTDs. Another class is that of *relational DTDs*. We say that D is a relational DTD if for each XML tree $T \models D$, if X is a non-empty subset of $tuples_D(T)$, then $trees_D(X) \models D$. This class contains regular expressions like the one below, from a DTD for Frequently Asked Questions [HJ99]:

```
<!ELEMENT section (logo*, title, (qna+ | q+ | ( p | div | section)+))>
```

There exist non-relational DTDs (for example, `<!ELEMENT a (b,b)>`). However:

Proposition 6.3.4 *Every disjunctive DTD is relational.*

Proof: Let $D = (E, A, P, R, r)$ be a disjunctive DTD, T an XML tree conforming to D and X a non-empty subset of $tuples_D(T)$. Assume that $trees_D(X) \not\models D$, that is, there is an XML tree $T' = (V, lab, ele, att, root)$ in $trees_D(X)$ such that $T' \not\models D$. Then, there is a vertex $v \in V$ reachable from the root by following a path p such that $lab(v) = \tau$ and $ele(v)$ does not conform to the regular expression $P(\tau)$.

If $P(\tau) = s$, where s is a simple disjunction over an alphabet A, then there is $t' \in X$ such that $t'.p = v$ and $t'.p.a = \perp$, for each $a \in A$. Thus, given that $T \models D$, we conclude that there is a tuple $t \in tuples_D(T)$ such that $t.p.b \neq \perp$, for some $b \in A$, and $t'.w = t.w$ for each $w \in paths(D)$ such that $p.b$ is not a prefix of w. Hence, $t' \sqsubset t$. But, this contradicts the definition of $tuples_D(\cdot)$, since $t', t \in tuples_D(T)$. The proof for $P(\tau) = s_1, \ldots, s_n$, where each s_i ($i \in [1, n]$) is either a simple regular expression or a simple disjunction, is similar. □

Theorem 6.3.5 *The FD implication problem over relational DTDs and over disjunctive DTDs is coNP-complete.*

PROOF: Here we prove the intractability of the implication problem for disjunctive DTDs. The coNP membership proof can be found in Appendix C.4.

In order to prove the coNP-hardness, we will reduce SAT-CNF to the complement of the implication problem for disjunctive DTDs. Let θ be a propositional formula of the form $C_1 \wedge \cdots \wedge C_n$, where each C_i ($i \in [1, n]$) is a clause. Assume that θ uses propositional variables $x_1, \ldots, x_m$.

We need to construct a disjunctive DTD D and a set of functional dependencies $\Sigma \cup \{\varphi\}$ such that $(D, \Sigma) \not\vdash \varphi$ if and only if θ is satisfiable. We define the DTD $D = (E, A, P, R, r)$ as follows.

$$
\begin{aligned}
E &= \{r, B, C\} \cup \{C_{i,j} \mid C_i \text{ mentions literal } x_j\} \cup \{\bar{C}_{i,j} \mid C_i \text{ mentions literal } \neg x_j\},\\
A &= \{@l\}.
\end{aligned}
$$

In order to define P, first we define a function for translating clauses into regular expressions. If the set of literal mentioned in the clause C_i ($i \in [1, n]$) is $\{x_{j_1}, \ldots, x_{j_p}, \bar{x}_{k_1}, \ldots, \bar{x}_{k_q}\}$, then

$$
tr(C_i) \;=\; C_{i,j_1}|\cdots|C_{i,j_p}|\bar{C}_{i,k_1}|\cdots|\bar{C}_{i,k_q}.
$$

We define the function P on the root as $P(r) = tr(C_1), \ldots, tr(C_n), B, C^*$. For the remaining elements of E, we define P as ϵ. Finally, $R(r) = \emptyset$ and $R(\tau) = \{@l\}$ for every $\tau \in E - \{r\}$. For example, figure 6.1 shows the DTD generated from a propositional formula $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_3)$.

For each pair of elements $C_{i,j}, \bar{C}_{k,j} \in E$, the set of functional dependencies Σ includes the constraint $\{r.C_{i,j}.@l,\ r.\bar{C}_{k,j}.@l\} \to r.C.@l$. Functional dependency φ is defined as $r.B.@l \to r.C.@l$.

We now prove that $(D, \Sigma) \not\vdash \varphi$ if and only if θ is satisfiable.

($\Rightarrow$) Suppose that $(D, \Sigma) \not\vdash \varphi$. Then, there is an XML tree T such that $T \models (D, \Sigma)$

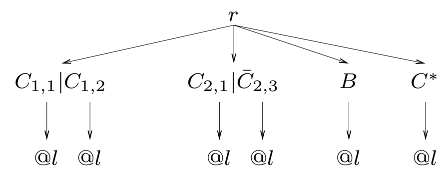


Figure 6.1: DTD generated from a formula $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_3)$.

and $T \not\models \varphi$. We define a truth assignment σ from T as follows. For each $j \in [1, m]$, if there is $i \in [1, n]$ such that r has a child of type $C_{i,j}$ in T, then $\sigma(x_j) = 1$, otherwise $\sigma(x_j) = 0$. We now prove that $\sigma \models C_i$, for each $i \in [1, n]$. By definition of D, there is $j \in [1, m]$ such that r has a child in T of type either $C_{i,j}$ or $\bar{C}_{i,j}$. In the first case, C_i contains the literal x_j and $\sigma(x_j) = 1$, by definition of σ. Therefore, $\sigma \models C_i$. In the second case, C_i contains a literal $\neg x_j$. If $\sigma(x_j) = 1$, then there is $k \in [1, n]$ such that r has a child of type $C_{k,j}$ in T, by definition of σ. Since $\{r.C_{k,j}.@l,\ r.\bar{C}_{i,j}.@l\} \to r.C.@l$ is a constraint in Σ, all the nodes in T of type C have the same value in the attribute $@l$. Thus, $T \models r.B.@l \to r.C.@l$, a contradiction. Hence, $\sigma(x_j) = 0$ and $\sigma \models C_i$.

($\Leftarrow$) Suppose that θ is satisfiable. Then, there exists a truth assignment σ such that $\sigma \models \theta$. We define an XML tree T conforming to D as follows. For each $i \in [1, n]$, choose a literal l_j in C_i such that $\sigma \models l_j$. If $l_j = x_j$, then r has a child of type $C_{i,j}$ in T, otherwise r has a child of type $\bar{C}_{i,j}$ in T. Moreover, r has one child of type B and two children of type C. We assign two distinct values to the attribute $@l$ of the nodes of type C, and the same value to the rest of the attributes in T. Thus, $T \not\models \varphi$, and it is easy to verify that $T \models \Sigma$. This completes the proof. $\square$

6.3.5 Nonaxiomatizability of XML functional dependencies

In this section we present a simple proof that XML FDs cannot be finitely axiomatized. This proof shows that, unlike relational databases, there is a nontrivial interaction between DTDs and functional dependencies. To present this proof we need to introduce some terminology.

Given a DTD D and a set of functional dependencies Σ over D, we say that (D, Σ) is *closed under implication* if for every FD φ over D such that $(D, \Sigma) \vdash \varphi$, it is the case that $\varphi \in \Sigma$. Furthermore, we say that (D, Σ) is *closed under k-ary implication*, $k \geq 0$, if for every FD φ over D, if there exists $\Sigma' \subseteq \Sigma$ such that $|\Sigma'| \leq k$ and $(D, \Sigma') \vdash \varphi$, then

$\varphi \in \Sigma$. For example, if (D, Σ) is closed under 0-ary implication, then Σ contains all the trivial FDs.

Since the implication problem for relational FDs is finitely axiomatizable, there exists $k \geq 0$ such that each relation schema $R(A_1, \ldots, A_n)$ admits a k-ary ground axiomatization for the implication problem, that is, an axiomatization containing rules of the form *if* Γ *then* γ, where $\Gamma \cup \{\gamma\}$ is a set of FDs over $R(A_1, \ldots, A_n)$ and $|\Gamma| \leq k$. For instance, $R(A, B, C)$ admits a 2-ary ground axiomatization including, among others, the following rules: *if* $\emptyset$ *then* $AB \to A$, *if* $A \to B$ *then* $AC \to BC$ and *if* $\{A \to B, B \to C\}$ *then* $A \to C$. Similarly, if there exists a finite axiomatization for the implication problem of XML FDs, then there exists $k \geq 0$ such that each DTD D admits a (possible infinite) k-ary ground axiomatization for the implication problem. The contrapositive of the following proposition gives us a sufficient condition for showing that the XML FD implication problem does not admit a k-ary ground axiomatization for every $k \geq 0$ and, therefore, it does not admit a finite axiomatization.

Proposition 6.3.6 *For every $k \geq 0$, if there is a k-ary ground axiomatization for the implication problem of XML FDs, then for every DTD D and set of FDs Σ over D, if (D, Σ) is closed under k-ary implication then (D, Σ) is closed under implication.*

PROOF: This proposition was proved in [AHV95] for the case of relational databases. The proof for XML FDs is similar. □

Theorem 6.3.7 *The implication problem for XML functional dependencies is not finitely axiomatizable.*

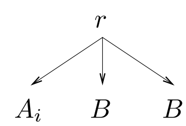
PROOF: By Proposition 6.3.6, for every $k \geq 0$ we need to exhibit a DTD D_k and a set of functional dependencies Σ_k such that (D_k, Σ_k) is closed under k-ary implication and (D_k, Σ_k) is not closed under implication.

The DTD $D_k = (E, A, P, R, r)$ is defined as follows: $E = \{A_1, \ldots, A_k, A_{k+1}, B\}$, $A = \emptyset$, $P(r) = (A_1| \cdots |A_k|A_{k+1}), B^*$ and $P(\tau) = \epsilon$ for every $\tau \in E - \{r\}$. The set of FDs Σ_k is defined as the union of the following sets:

- $\{r.A_i \to r.B \mid i \in [1, k+1]\} \cup \{\{r, r.A_i\} \to r.B \mid i \in [1, k+1]\}$,
- $\{S \to p \mid S \to p$ is a trivial FD in $D_k\}$.

It is easy to see that if φ is not a trivial functional dependency in D_k and $\varphi \notin \Sigma_k$, then $\varphi = r \to r.B$. Thus, in order to prove that (D_k, Σ_k) is closed under k-ary implication and is not closed under implication, we have to show that:

1. For every $\Sigma' \subseteq \Sigma_k$ such that $|\Sigma'| \leq k$, $(D_k, \Sigma') \not\vdash r \to r.B$. Since $|\Sigma'| \leq k$, there exists $i \in [1, k+1]$ such that $r.A_i \to r.B \notin \Sigma'$ and $\{r, r.A_i\} \to r.B \notin \Sigma'$. Thus, an XML tree T defined as

   conforms to D_k, satisfies Σ' and does not satisfy $r \to r.B$. We conclude that $(D_k, \Sigma') \not\vdash r \to r.B$.

2. $(D_k, \Sigma_k) \vdash r \to r.B$. This proof is straightforward.

This completes the proof of the theorem. □

6.4 The Consistency Problem for XML Functional Dependencies

As we mentioned in the previous chapters, an XML specification can be inconsistent in the sense that there is no way of populating the database and satisfying both the DTD and the set of data dependencies given by the specification. In particular, XML databases containing functional dependencies can be inconsistent.

Inconsistent XML databases are poorly designed and, thus, it is desirable to have algorithms for checking consistency. In Chapter 5, we study the complexity of checking consistency for XML databases containing keys and foreign keys. In this section, we study the complexity of the consistency problem for XML functional dependencies.

We start by noticing that if a relational DTD D is consistent, then there exists an XML tree T conforming to D and containing only one tree tuple ($|tuples_D(T)| = 1$). Thus, if D is consistent, then for every set Σ of functional dependencies over D, we have that (D, Σ) is consistent since T trivially satisfies every functional dependency of Σ. Hence, given a relational DTD D and a set Σ of FDs over D, there exists an XML document conforming to D and satisfying Σ if and only if there exists an XML document conforming to D. Thus, the consistency problem for relational DTDs and FDs can be reduced to the consistency problem for DTDs, which in turn can be reduced in linear time to the emptiness problem for context free grammars [FL02]. Since the latter problem can be solved in linear time (cf. [HU79]), the following theorem is obtained.

Theorem 6.4.1 *The consistency problem for XML functional dependencies over relational DTDs is decidable in linear time.*

Given that that simple (disjunctive) DTDs are also relational, the following corollary is obtained.

Corollary 6.4.2 *The consistency problem for XML functional dependencies over simple (disjunctive) DTDs is decidable in linear time.*

To obtain the decidability of the consistency problem for functional dependencies, we reduce this problem to the implication problem for this class of constraints.

Proposition 6.4.3 *Given a DTD D and set Σ of FDs over D, it is possible to construct in linear time a DTD D' and an FD φ such that (D, Σ) is consistent iff $(D', \Sigma) \not\vdash \varphi$.*

Proof: Assume that $D = (E, A, P, R, r)$. Then define $D' = (E', A, P', R, r)$ as follows. The new set of element types E' is defined to be $E \cup \{a, b\}$, where a and b are fresh element types, and function P' is defined as $P'(r) = P(r), a, b^*$, $P(a) = P(b) = \epsilon$ and $P'(\tau) = P(\tau)$ for every $\tau \in E - \{r, a, b\}$.

Furthermore, define functional dependency φ as $r.a \to r.b$. Then it is easy to see that (D, Σ) is consistent if and only if $(D', \Sigma) \not\vdash \varphi$. □

From Proposition 6.4.3 and Theorem 6.3.1, we obtain the decidability of the consistency problem for XML functional dependencies. A slight modification of the proof of coNP-hardness of the implication problem over relational DTDs (Theorem 6.3.5) shows that the consistency problem for XML FDs is NP-hard.[2]

Theorem 6.4.4 *The consistency problem for XML functional dependencies over DTDs is NP-hard, and can be solved in NEXPTIME.*

[2]To prove this lower bound, we are forced to transform the relational DTD used in the proof of Theorem 6.3.5 into a non-relational DTD.

6.5 Related Work

Fan and Simeón [FS00, FS03] introduced a simple language for expressing functional dependencies for XML. In this language, a functional dependency is a constraint of the form $\tau.p \rightarrow \tau.q$, where τ is an element type and p, q are paths. An XML tree T satisfies this constraint if for every pair of nodes x, y in T of type τ, if $\mathit{reach}(x, p) = \mathit{reach}(y, p)$, then $\mathit{reach}(x, q) = \mathit{reach}(y, q)$. This language only allows unary functional dependencies that hold in the entire document (absolute constraints). Lee et al. [LLL02] also introduced a language for expressing functional dependencies for XML. In that language, a functional dependency is an expression of the form $(p, [q_1, \ldots, q_n \rightarrow q_{n+1}])$, where p is a path and every q_i ($i \in [1, n+1]$) is of the form $\tau.@l$, where τ is an element type and $@l$ is an attribute. An XML tree T satisfies this constraint if for any two subtrees T_1, T_2 of T whose roots are nodes reachable from the root of T by following path p, if T_1 and T_2 agree on the value of q_i, for every $i \in [1, n]$, then they agree on the value of q_{n+1}. This language does not consider relative constraints and its semantics only works properly if some syntactic restrictions are imposed on the functional dependencies [LLL02].

Chapter 7

XNF: A Normal Form for XML Documents

We have developed all the elements that we need to introduce a normal form for XML documents: a set of information-theoretic tools for testing when a normal form corresponds to a good design, a formal model for XML databases and a functional dependency language for XML. Thus, in this chapter, we finally propose a normal form for XML documents. More specifically, we show that, like relational databases, XML documents may contain redundant information, and may be prone to update anomalies. We define an XML normal form, XNF, that avoids update anomalies and redundancies. We study its properties, and show that the information-theoretic measure introduced in Chapter 3 justifies XNF. We also show that XNF generalizes BCNF, and we discuss the relationship between XNF and normal forms for nested relations. Finally, we present a lossless algorithm for converting any DTD into one in XNF, and we look at information-theoretic criteria for justifying this algorithm.

7.1 Introduction

The concepts of database design and normal forms are a key component of the relational database technology. In this chapter, we study design principles for XML data. XML has recently emerged as a new basic format for data exchange. Although many XML documents are views of relational data, the number of applications using native XML documents is increasing rapidly. Such applications may use native XML storage facilities [KM00], and update XML data [TIHW01]. Updates, like in relational databases, may cause anomalies if data is redundant. In the relational world, anomalies are avoided by using well-designed database schema. XML has its version of schema too; most often it is DTDs (Document Type Definitions), and some other proposals exist or are under development [TBMM, LJM+]. What would it mean then for such a schema to be well or poorly designed? Clearly, this question has arisen in practice: one can find companies offering help in "good DTD design." This help, however, comes in form of consulting services rather than commercially available software, as there are no clear guidelines for producing well designed XML.

Our goal is to find principles for good XML data design, and algorithms to produce such designs. We believe that it is important to do this research now, as a lot of data is being put on the web. Once massive web databases are created, it is very hard to change their organization; thus, there is a risk of having large amounts of widely accessible, but at the same time poorly organized legacy data.

Normalization is one of the most thoroughly researched subjects in database theory. Here we follow the standard treatment of one of the most common (if not the most common) normal forms, BCNF. It eliminates redundancies and avoids update anomalies (see Section 2.1.4) which they cause by decomposing into relational subschemas in which every nontrivial functional dependency defines a key. Just to retrace this development in the XML context, we need the following:

a) Understanding of what a redundancy and an update anomaly is.

b) A definition and basic properties of functional dependencies (so far, most proposals for XML constraints concentrate on keys).

c) A definition of what "bad" functional dependencies are (those that cause redundancies and update anomalies).

d) An algorithm for converting an arbitrary DTD into one that does not admit such bad functional dependencies.

Starting with point a), how does one identify bad designs? We have looked at a large number of DTDs and found two kinds of commonly present design problems. They are illustrated in two examples below.

Example 7.1.1 Consider the following DTD that describes a part of a university database:

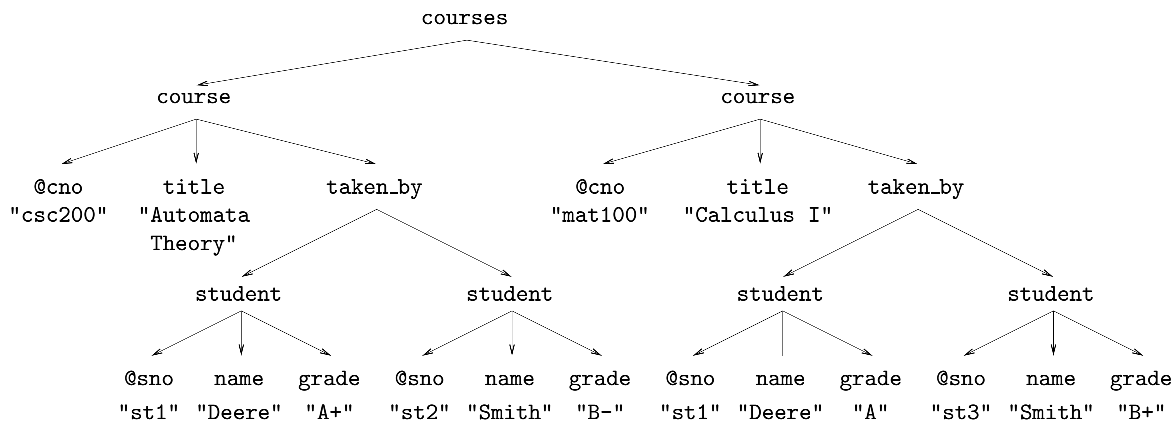


Figure 7.1: A document containing redundant information.

```
<!DOCTYPE courses [
  <!ELEMENT courses (course*)>
  <!ELEMENT course (title, taken_by)>
    <!ATTLIST course
              cno CDATA #REQUIRED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT taken_by (student*)>
  <!ELEMENT student (name, grade)>
    <!ATTLIST student
              sno CDATA #REQUIRED>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT grade (#PCDATA)>
]>
```

For every course, we store its number (`cno`), its title and the list of students taking the course. For each student taking a course, we store his/her number (`sno`), name, and the grade in the course.

An example of an XML document that conforms to this DTD is shown in Figure 7.1. This document satisfies the following constraint: any two `student` elements with the same `sno` value must have the same `name`. This constraint (which looks very much like a functional dependency), causes the document to store redundant information: for example, the name `Deere` for student `st1` is stored twice. And just as in relational databases, such redundancies can lead to update anomalies: for example, updating the name of `st1` for only one course results in an inconsistent document, and removing the student from a course may result in removing that student from the document altogether.

In order to eliminate redundant information, we use a technique similar to the relational one, and split the information about the name and the grade. Since we deal with just one XML document, we must do it by creating an extra element type, `info`, for student information, as shown below:

```
<!DOCTYPE courses [
  <!ELEMENT courses (course*, info*)>
  <!ELEMENT course (title,taken_by)>
    <!ATTLIST course
              cno CDATA #REQUIRED>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT taken_by (student*)>
  <!ELEMENT student (grade)>
    <!ATTLIST student
              sno CDATA #REQUIRED>
  <!ELEMENT grade (#PCDATA)>
  <!ELEMENT info (number*,name)>
  <!ELEMENT number EMPTY>
    <!ATTLIST number
              sno CDATA #REQUIRED>
  <!ELEMENT name (#PCDATA)>
]>
```

Each `info` element has as children one `name` and a sequence of `number` elements, with `sno` as an attribute. Different students can have the same name, and we group all student numbers `sno` for each `name` under the same `info` element. A restructured document that conforms to this DTD is shown in Figure 7.2. Note that `st2` and `st3` are put together because both students have the same name. □

This example is reminiscent of the canonical example of bad relational design caused by non-key functional dependencies, and so is the modification of the schema. Some examples of redundancies are more closely related to the hierarchical structure of XML documents.

Example 7.1.2 The DTD below is a part of the DBLP database [Ley] for storing data about conferences.

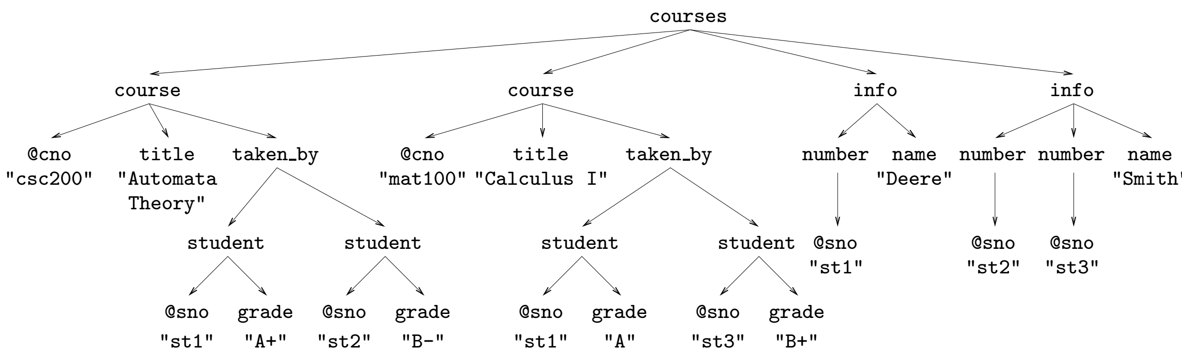


Figure 7.2: A well-designed document.

```
<!DOCTYPE db [
  <!ELEMENT db (conf*)>
  <!ELEMENT conf (title, issue+)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT issue (inproceedings+)>
  <!ELEMENT inproceedings (author+, title)>
    <!ATTLIST inproceedings
              key ID #REQUIRED
              pages CDATA #REQUIRED
              year CDATA #REQUIRED>
  <!ELEMENT author (#PCDATA)>
]>
```

Each conference has a title, and one or more issues (which correspond to years when the conference was held). Papers are stored in `inproceedings` elements; the year of publication is one of its attributes.

Such a document satisfies the following constraint: any two `inproceedings` children of the same `issue` must have the same value of `year`. This too is similar to relational functional dependencies, but now we refer to the values (the `year` attribute) as well as the structure (children of the same `issue`). Moreover, we only talk about `inproceedings` nodes that are children of the same `issue` element. Thus, this functional dependency can be considered relative to each `issue`.

The functional dependency here leads to redundancy: year is stored multiple times for a conference. The natural solution to the problem in this case is not to create a new element for storing the year, but rather restructure the document and make `year` an attribute of `issue`. That is, we change attribute lists as:

```
<!ATTLIST issue
          year CDATA #REQUIRED>
<!ATTLIST inproceedings
          key ID #REQUIRED
          pages CDATA #REQUIRED>
```

□

Our goal is to show how to detect anomalies of those kinds, and to transform documents in a lossless fashion into ones that do not suffer from those problems.

The first step towards that goal is to introduce functional dependencies (FDs) for XML documents. We already achieved this goal in Chapter 6, where we introduced FDs for XML by considering a relational representation of documents, via tree tuples, and defining FDs on them.

The second step is the definition of a normal form that disallows redundancy-causing FDs. We give it in Section 7.2, and show that our normal form, called XNF, generalizes BCNF and a nested normal form NNF-96 [MNE96] when only functional dependencies are considered. In Section 7.3 we study the complexity of testing XNF.

The third step is to formally justify XNF. We do this in Section 7.4, where we show that the information-theoretic measure of Chapter 3 straightforwardly extends to the XML setting, giving us a definition of well-designed XML specifications. We prove that for constraints given by XML functional dependencies, well-designed XML specifications are precisely those in XNF.

The last step then is to find an algorithm that converts any DTD, given a set of FDs, into one in XNF. We do this in Section 7.5. On both examples shown earlier, the algorithm produces exactly the desired reconstruction of the DTD. The main algorithm uses implication of functional dependencies (although there is a version that does not use implication, but it may produce suboptimal results). It is worth mentioning that in Section 7.5.3, we use the information-theoretic measure of Section 7.4 to show that the algorithm do not decrease the information content of each datum at every step, which is the criterion used in Chapter 3 to test whether a relational normalization algorithm is good. Finally, in Section 7.7 we describe related work.

One of the reasons for the success of the normalization theory is its simplicity, at least for the commonly used normal forms such as BCNF, 3NF and 4NF. Hence, the normalization theory for XML should not be extremely complicated in order to be applicable. In particular, this was the reason we chose to use DTDs instead of more complex formalisms [TBMM]. This is in perfect analogy with the situation in the relational world: although SQL DDL is a rather complicated language with numerous features, BCNF decomposition uses a simple model of a set of attributes and a set of functional dependencies.

7.2 XNF: An XML Normal Form

With the definitions of Chapter 6, we are ready to present the normal form that generalizes BCNF for XML documents.

Definition 7.2.1 *Given a DTD D and $\Sigma \subseteq \mathcal{FD}(D)$, (D, Σ)* is in XML normal form (XNF) *iff for every nontrivial FD $\varphi \in (D, \Sigma)^+$ of the form $S \rightarrow p.@l$ or $S \rightarrow p.\mathtt{S}$, it is the case that $S \rightarrow p$ is in $(D, \Sigma)^+$.*

The intuition is as follows. Suppose that $S \rightarrow p.@l$ is in $(D, \Sigma)^+$. If T is an XML tree conforming to D and satisfying Σ, then in T for every set of values of the elements in S, we can find only one value of $p.@l$. Thus, for every set of values of S we need to store the value of $p.@l$ only once; in other words, $S \rightarrow p$ must be implied by (D, Σ).

In this definition, we impose the condition that φ is a nontrivial FD. Indeed, the trivial FD $p.@l \rightarrow p.@l$ is always in $(D, \Sigma)^+$, but often $p.@l \rightarrow p \notin (D, \Sigma)^+$, which does not necessarily represent a bad design.

To show how XNF distinguishes good XML design from bad design, we revisit the examples from the introduction, and prove that XNF generalizes BCNF and NNF-96, a normal form for nested relations [MNE96], when only functional dependencies are provided.

Example 7.2.2 Referring back to example 7.1.1, we have the following FDs. `cno` is a key of `course`:

$$courses.course.@cno \rightarrow courses.course. \tag{FD1}$$

Another FD says that two distinct `student` subelements of the same `course` cannot have the same `sno`:

$$\{courses.course,\ courses.course.taken\_by.student.@sno\} \rightarrow$$
$$courses.course.taken\_by.student. \quad \text{(FD2)}$$

Finally, to say that two `student` elements with the same `sno` value must have the same `name`, we use

$$courses.course.taken\_by.student.@sno \rightarrow$$
$$courses.course.taken\_by.student.name.\mathtt{S}. \quad \text{(FD3)}$$

Functional dependency (FD3) associates a unique name with each student number, which is therefore redundant. The design is *not* in XNF, since it contains (FD3) but does not imply the functional dependency

$$courses.course.taken\_by.student.@sno \quad \rightarrow \quad courses.course.taken\_by.student.name.$$

To remedy this, we gave a revised DTD in example 7.1.1. The idea was to create a new element `info` for storing information about students. That design satisfies FDs (FD1), (FD2) as well as

$$courses.info.number.@sno \quad \rightarrow \quad courses.info,$$

and can be easily verified to be in XNF. □

Example 7.2.3 Suppose that D is the DBLP DTD from example 7.1.2. Among the set Σ of FDs satisfied by the documents are:

$$db.conf.title.\mathtt{S} \rightarrow db.conf \quad \text{(FD4)}$$
$$db.conf.issue \rightarrow db.conf.issue.inproceedings.@year \quad \text{(FD5)}$$
$$\{db.conf.issue,\ db.conf.issue.inproceedings.title.\mathtt{S}\} \rightarrow$$
$$db.conf.issue.inproceedings \quad \text{(FD6)}$$
$$db.conf.issue.inproceedings.@key \rightarrow db.conf.issue.inproceedings \quad \text{(FD7)}$$

Constraint (FD4) enforces that two distinct conferences have distinct titles. Given that an issue of a conference represents a particular year of the conference, constraint (FD5) enforces that two articles of the same issue must have the same value in the attribute year. Constraint (FD6) enforces that for a given issue of a conference, two distinct articles must have different titles. Finally, constraint (FD7) enforces that `key` is an identifier for each article in the database.

By (FD5) for each issue of a conference, its year is stored in every article in that issue and, thus, DBLP documents can store redundant information. (D, Σ) is not in XNF, since

$$db.conf.issue \to db.conf.issue.inproceedings$$

is not in $(D, \Sigma)^+$.

The solution we proposed in the introduction was to make `year` an attribute of `issue`. (FD5) is not valid in the revised specification, which can be easily verified to be in XNF. Note that we do not replace (FD5) by $db.conf.issue \to db.conf.issue.@year$, since it is a trivial FD and thus is implied by the new DTD alone. □

7.2.1 BCNF and XNF

Relational databases can be easily mapped into XML documents. Given a relation $G(A_1, \ldots, A_n)$ and a set of FDs FD over G, we translate the schema (G, FD) into an XML representation, that is, a DTD and a set of XML FDs (D_G, Σ_{FD}). The DTD $D_G = (E,\ A,\ P,\ R,\ db)$ is defined as follows:

- $E = \{db, G\}$.
- $A = \{@A_1, \ldots, @A_n\}$.
- $P(db) = G^*$ and $P(G) = \epsilon$.
- $R(db) = \emptyset$, $R(G) = \{@A_1, \ldots, @A_n\}$.

Without loss of generality, assume that all FDs are of the form $X \to A$, where A is an attribute. Then Σ_{FD} over D_G is defined as follows.

- For each FD $A_{i_1} \cdots A_{i_m} \to A_i \in FD$, $\{db.G.@A_{i_1},\ \ldots,\ db.G.@A_{i_m}\} \to db.G.@A_i$ is in Σ_{FD}.
- $\{db.G.@A_1,\ \ldots,\ db.G.@A_n\} \to db.G$ is in Σ_{FD}.

The latter is included to avoid duplicates.

Example 7.2.4 A schema $G(A, B, C)$ can be coded by the following DTD:

```
<!ELEMENT db (G*)>
<!ELEMENT G EMPTY>
  <!ATTLIST G
            A CDATA #REQUIRED
            B CDATA #REQUIRED
            C CDATA #REQUIRED>
```

In this schema, an FD $A \to B$ is translated into $db.G.@A \to db.G.@B$. □

The following proposition shows that BCNF and XNF are equivalent when relational databases are appropriately coded as XML documents.

Proposition 7.2.5 *Given a relation schema $G(A_1, \ldots, A_n)$ and a set of functional dependencies FD over G, (G, FD) is in BCNF iff (D_G, Σ_{FD}) is in XNF.*

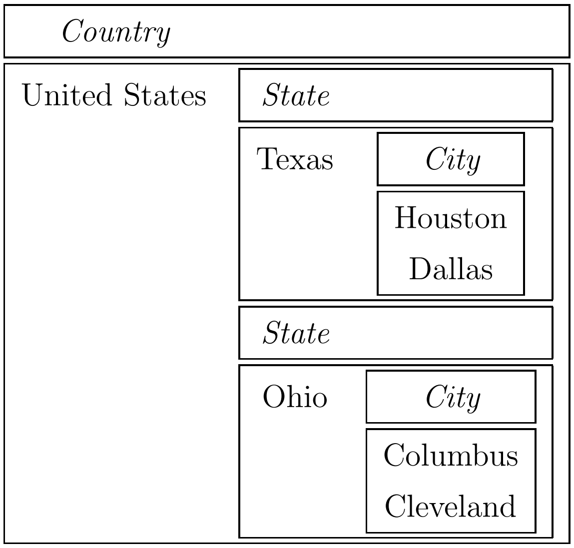
Proof: This follows from Proposition 7.2.6 (to be proved in the next section) since every relation schema is trivially consistent (see next section) and NNF-FD coincides with BCNF when only functional dependencies are provided [MNE96]. □

7.2.2 NNF-96 and XNF

In this section we assume familiarity with the terminology introduced in Section 2.2. Recall that a nested relation schema is either a set of attributes X, or $X(G_1)^* \ldots (G_n)^*$, where G_i's are nested relation schemas. An example of a nested relation for the schema $H_1 = Country(H_2)^*$, $H_2 = State(H_3)^*$, $H_3 = City$ is shown in Figure 7.3 (a).

Nested schemas are naturally mapped into DTDs, as they are defined by means of regular expressions. For a nested schema $G = X(G_1)^* \ldots (G_n)^*$, we introduce an element type G with $P(G) = G_1^*, \ldots, G_n^*$ and $R(G) = \{@A_1, \ldots, @A_m\}$, where $X = \{A_1, \ldots, A_m\}$; at the top level we have a new element type db with $P(db) = G^*$ and $R(db) = \emptyset$. In our example the DTD is:

```
<!DOCTYPE db [
  <!ELEMENT db (H1*)>
```



(a) Nested relation I_1

Country	*State*	*City*
United States	Texas	Houston
United States	Texas	Dallas
United States	Ohio	Columbus
United States	Ohio	Cleveland

(b) Complete unnesting of I_1

Figure 7.3: Nested relation and its unnesting.

```
  <!ELEMENT H1 (H2*)>
    <!ATTLIST H1 Country CDATA #REQUIRED>
  <!ELEMENT H2 (H3*)>
    <!ATTLIST H2 State CDATA #REQUIRED>
  <!ELEMENT H3 EMPTY>
    <!ATTLIST H3 City CDATA #REQUIRED>
]>
```

In [MNE96, ÖY87], functional dependencies are defined by following the flat approach presented in Section 2.2.1, that is, a functional dependency holds in a nested relation I if and only if it holds in the total unnesting of I. Thus, for example, nested relation I_1 shown in Figure 7.3 satisfies FD *State* $\rightarrow$ *Country* since its total unnesting, shown in Figure 7.3 (b), satisfies this constraint. On the other hand, FD *State* $\rightarrow$ *City* does not hold in I_1.

Normalization is usually considered for nested relations in the partition normal form (PNF). Note that PNF can be enforced by using FDs on the XML representation. In our example this is done as follows:

$$\begin{aligned} db.H_1.@\mathit{Country} &\rightarrow db.H_1 \\ \{db.H_1,\ db.H_1.H_2.@\mathit{State}\} &\rightarrow db.H_1.H_2 \\ \{db.H_1.H_2,\ db.H_1.H_2.H_3.@\mathit{City}\} &\rightarrow db.H_1.H_2.H_3 \end{aligned}$$

It turns out that one can define FDs over nested relations by using the XML representation. Let U be a set of attributes, G_1 a nested relation schema over U and FD a set of functional dependencies over G_1. Assume that G_1 includes nested relation schemas G_2, $\ldots$, G_n and a set of attributes $U' \subseteq U$. For each G_i ($i \in [1,n]$), $\mathit{path}(G_i)$ is inductively defined as follows. If $G_i = G_1$, then $\mathit{path}(G_i) = db.G_1$. Otherwise, if G_i is a nested attribute of G_j, then $\mathit{path}(G_i) = \mathit{path}(G_j).G_i$. Furthermore, if $A \in U'$ is an atomic attribute of G_i, then $\mathit{path}(A) = \mathit{path}(G_i).@A$. For instance, for the schema of the nested relation in Figure 7.3 (a), $\mathit{path}(H_2) = db.H_1.H_2$ and $\mathit{path}(\mathit{City}) = db.H_1.H_2.H_3.@\mathit{City}$.

We now define Σ_{FD} as follows:

- For each FD $A_{i_1} \cdots A_{i_m} \rightarrow A_i \in FD$, $\{\mathit{path}(A_{i_1}),\ \ldots,\ \mathit{path}(A_{i_m})\} \rightarrow \mathit{path}(A_i)$ is in Σ_{FD}.

- For each $i \in [1,n]$, if $A_{j_1}, \ldots, A_{j_m}$ is the set of atomic attributes of G_i and G_i is a nested attribute of G_j, $\{\mathit{path}(G_j),\ \mathit{path}(A_{j_1}),\ \ldots,\ \mathit{path}(A_{j_m})\} \rightarrow \mathit{path}(G_i)$ is in Σ_{FD}.

  Furthermore, if $B_{j_1}, \ldots, B_{j_l}$ is the set of atomic attributes of G_1, then $\{\mathit{path}(B_{j_1}),\ \ldots,\ \mathit{path}(B_{j_l})\} \rightarrow \mathit{path}(G_1)$ is in Σ_{FD}.

Note that the last rule imposes the partition normal form. The set Σ_{PNF} contains all the functional dependencies defined by this rule.

In Section 2.2.2, we introduced NNF-96 [MNE96]. This normal form was defined for nested schemas containing functional and multivalued dependencies. Here we consider a normal form NNF-FD, which is NNF-96 restricted to FDs only. Recall that $\mathit{MVD}(G)$, with G being a nested schema, stands for the set of multivalued dependencies embedded in G. For example, if $G_1 = \mathit{Title}(G_2)^*(G_3)^*$, $G_2 = \mathit{Director}$, $G_3 = \mathit{Theater}(G_4)^*$, $G_4 = \mathit{Snack}$, then $\mathit{MVD}(G_1)$ is equal to

$$\{\mathit{Title} \twoheadrightarrow \mathit{Director},\ \mathit{Title} \twoheadrightarrow \{\mathit{Theater}, \mathit{Snack}\},\ \{\mathit{Title}, \mathit{Theater}\} \twoheadrightarrow \mathit{Snack}\},$$

Given a nested relation schema G and a set FD of functional dependencies over G, we say that (G, FD) is in NNF-FD if (1) $FD \vdash \mathit{MVD}(G)$, that is, every multivalued dependency embedded in G is implied by FD, and (2) for each nontrivial FD $X \to A \in FD^+$, $X \to \mathit{Ancestor}(N_A)$ is also in FD^+, where N_A is the node in the schema tree of G that contains attribute A. As in Section 2.2, FD^+ stands for the set of all FDs implied by FD.

The following proposition shows that NNF-FD and XNF are equivalent when nested relational databases are appropriately coded as XML documents. Recall that (G, FD) is consistent [MNE96] if $FD \vdash MVD(G)$.

Proposition 7.2.6 *Let G be a nested relation schema and FD a set of functional dependencies over G such that (G, FD) is consistent. Then (G, FD) is in NNF-FD iff (D_G, Σ_{FD}) is in XNF.*

PROOF: First we need to prove the following claim.

Claim 7.2.7 $A_{i_1} \cdots A_{i_m} \to A_i \in FD^+$ *if and only if* $\{\mathit{path}(A_{i_1}), \ldots, \mathit{path}(A_{i_m})\} \to \mathit{path}(A_i) \in (D_G, \Sigma_{FD})^+$.

The proof of this claim follows from the following fact. For each instance I of G, there is an XML tree T_I conforming to D_G such that $I \models FD$ iff $T_I \models \Sigma_{FD}$. Moreover, for each XML tree T conforming to D_G and satisfying Σ_{PNF}, there is an instance I_T of G such that $T \models \Sigma_{FD}$ iff $I_T \models FD$.

Now we prove the proposition.

($\Leftarrow$) Suppose that (D_G, Σ_{FD}) is in XNF. We prove that (G, FD) is in NNF-FD. Given that (G, FD) is consistent, we only need to consider the second condition in the definition of NNF-FD. Let $A_{i_1} \cdots A_{i_m} \to A_i$ be a nontrivial functional dependency in FD^+. We have to prove that $A_{i_1}, \ldots, A_{i_m} \to \mathit{Ancestor}(N_{A_i})$ is in FD^+, where N_{A_i} is the node in the schema tree of G that contains attribute A_i. By Claim 7.2.7, we know that $\{\mathit{path}(A_{i_1}), \ldots, \mathit{path}(A_{i_m})\} \to \mathit{path}(A_i)$ is a nontrivial functional dependency in $(D_G, \Sigma_{FD})^+$. Since (D_G, Σ_{FD}) is in XNF, $\{\mathit{path}(A_{i_1}), \ldots, \mathit{path}(A_{i_m})\} \to \mathit{path}(G_j)$ is in $(D_G, \Sigma_{FD})^+$, where G_j is a nested relation schema contained in G such that A_i is an atomic attribute of G_j. Thus, given that $\mathit{path}(G_j) \to \mathit{path}(A)$ is a trivial functional dependency in D_G, for each $A \in \mathit{Ancestor}(N_{A_i})$, we conclude that $\{\mathit{path}(A_{i_1}), \ldots, \mathit{path}(A_{i_m})\} \to \mathit{path}(A)$ is in $(D_G, \Sigma_{FD})^+$ for each $A \in \mathit{Ancestor}(N_{A_i})$. By Claim 7.2.7, $A_{i_1} \cdots A_{i_m} \to \mathit{Ancestor}(N_{A_i})$ is in FD^+.

($\Rightarrow$) Suppose that (G, FD) is in NNF-FD. We will prove that (D_G, Σ_{FD}) is in XNF. Let R be a nested relation schema contained in G and A an atomic attribute of R. Suppose that there is $S \subseteq \mathit{paths}(D_G)$ such that $S \to \mathit{path}(A)$ is a nontrivial functional dependency in $(D_G, \Sigma_{FD})^+$. We have to prove that $S \to \mathit{path}(R) \in (D_G, \Sigma_{FD})^+$. Let S_1 and S_2 be set of paths such that $S = S_1 \cup S_2$, $S_1 \subseteq \mathit{EPaths}(D_G)$ and $S_2 \cap \mathit{EPaths}(D_G) = \emptyset$. Let $S_1' = \{\mathit{path}(A') \mid$ there is $\mathit{path}(R') \in S_1$ such that A' is an atomic attribute of some nested relation schema mentioned in $\mathit{path}(R')\}$. Given that $\Sigma_{PNF} \subseteq \Sigma_{FD}$, $S_1' \to S_1 \in (D_G, \Sigma_{FD})^+$. Thus, $S_1' \cup S_2 \to \mathit{path}(A) \in (D_G, \Sigma_{FD})^+$. Assume that $S_1' \cup S_2 = \{\mathit{path}(A_{i_1}), \ldots, \mathit{path}(A_{i_m})\}$. By Claim 7.2.7, $A_{i_1} \cdots A_{i_m} \to A$ is a nontrivial functional dependency in FD^+. Thus, given that (G, FD) is in NNF-FD, we conclude that $A_{i_1} \cdots A_{i_m} \to \mathit{Ancestor}(N_A)$ is in FD^+, where N_A is the node in the schema tree of G that contains attribute A. Therefore, by Claim 7.2.7, $S_1' \cup S_2 \to \mathit{path}(B)$ is in $(D_G, \Sigma_{FD})^+$, for each $B \in \mathit{Ancestor}(N_A)$. But $\{\mathit{path}(B) \mid B \in \mathit{Ancestor}(N_A)\} \to \mathit{path}(R)$ is in $(D_G, \Sigma_{FD})^+$, since $\Sigma_{PNF} \subseteq \Sigma_{FD}$. Thus, $S_1' \cup S_2 \to \mathit{path}(R) \in (D_G, \Sigma_{FD})^+$, and given that $S_1 \to S_1'$ is a trivial functional dependency in D_G, we conclude that $S \to \mathit{path}(R)$ is in $(D_G, \Sigma_{FD})^+$. This concludes the proof of the proposition. □

Finally, in the following example we show that in general XNF does not generalize NNF since it does not take into account multivalued dependencies.

Example 7.2.8 Let G_1 be nested schema $\mathit{Title}(G_2)^*(G_3)^*$, where $G_2 = \mathit{Director}$, $G_3 = \mathit{Theater}(G_4)^*$ and $G_4 = \mathit{Snack}$. Assume that Σ is the following set of multivalued dependencies:

$$\{\ \mathit{Title} \twoheadrightarrow \mathit{Director}, \quad \mathit{Title} \twoheadrightarrow \mathit{Theater}, \quad \mathit{Title} \twoheadrightarrow \mathit{Snack}\ \}.$$

Then (G, Σ) is not in NNF since the set of multivalued dependencies $\mathit{MVD}(G) = \{\mathit{Title} \twoheadrightarrow \mathit{Director}\}$ is not equivalent to Σ. On the other hand, the XML representation of (G, Σ) is trivially in XNF since Σ does not contain any functional dependency. □

7.3 The complexity of testing XNF

In Sections 4.2 and 6.3.4, we introduce simple DTDs and relational DTDs. In this section, we study the complexity of testing XNF for XML specifications containing these types of DTDs.

Relational DTDs have the following useful property that lets us establish the complexity of testing XNF.

Proposition 7.3.1 *Given a relational DTD D and a set Σ of FDs over D, (D, Σ) is in XNF iff for each nontrivial FD of the form $S \to p.@l$ or $S \to p.\mathtt{S}$ in Σ, $S \to p \in (D, \Sigma)^+$.*

PROOF: We only need to prove the "if" direction. Suppose that for each nontrivial FD of the form $S \to p.@l$ or $S \to p.\mathtt{S}$ in Σ, $S \to p \in (D, \Sigma)^+$.

Assume that (D, Σ) is not in XNF. Without loss of generality, assume that there exists a nontrivial functional dependency $S' \to p'.@l'$ such that $S' \to p'.@l' \in (D, \Sigma)^+$ and $S' \to p' \notin (D, \Sigma)^+$. By Lemma C.4.1, there is an XML tree T and a path q prefix of p' such that T conforms to D, T satisfies Σ, $\mathit{tuples}_D(T) = \{t_1, t_2\}$, $t_1.S' = t_2.S'$, $t_1.S' \neq \bot$, $t_1.p' \neq t_2.p'$, $t_1.q \neq t_2.q$ and for each $s \in \mathit{paths}(D)$, if q is not a prefix of s, then $t_1.s = t_2.s$. If $t_1.p'.@l' \neq t_2.p'.@l'$, then $(D, \Sigma) \not\vdash S' \to p'.@l'$, a contradiction. Thus, we can assume that $t_1.p'.@l' = t_2.p'.@l'$. We can also assume $t_1.p'.@l' \neq \bot$, since if $t_1.p'.@l' = t_2.p'.@l' = \bot$, then $t_1.p' = t_2.p' = \bot$ and, therefore, $T \models S' \to p'$. Define a new tree tuple t_1' as follows: $t_1'.w = t_1.w$, for each $w \neq p'.@l'$, $t_1'.p'.@l' \neq t_1.p'.@l'$ and $t_1'.p'.@l' \neq \bot$. Then, there is an XML tree $T' \in \mathit{trees}_D(\{t_1', t_2\})$ such that $T' \models D$ and $T' \not\models S' \to p'.@l'$, since $p'.@l' \notin S'$ ($S' \to p'.@l'$ is a nontrivial functional dependency). If $T' \models \Sigma$, then $(D, \Sigma) \not\vdash S' \to p'.@l'$, a contradiction. Hence $T' \not\models \Sigma$ and, therefore, there is $S \to p'' \in \Sigma$ such that $T' \not\models S \to p''$. But p'' must be equal to $p'.@l'$, since $t_1, t_2 \in \mathit{tuples}_D(T)$ and $T \models \Sigma$. Therefore, $T \not\models S \to p'$, because $t_1.S = t_1'.S = t_2.S$, $t_1'.S \neq \bot$ and $t_1.p' \neq t_2.p'$. We conclude that $(D, \Sigma) \not\vdash S \to p'$, which contradicts our initial assumption since $S \to p'.@l'$ is a nontrivial FD in Σ. □

From this and Theorems 6.3.2 and 6.3.5 we immediately derive:

Corollary 7.3.2 *Testing if (D, Σ) is in XNF can be done in cubic time for simple DTDs, and is coNP-complete for relational DTDs.*

7.4 Justifying XNF

In this section we show that the notion of being well-designed straightforwardly extends from relations to XML. Furthermore, if all constraints are specified as functional dependencies, this notion precisely characterizes XNF.

We do not need to introduce a new notion of being well-designed specifically for XML: the definition that we formulated in Section 3.4 for relational data will apply. We only have to define the notion of positions in a tree, and then reuse the relational definition. For relational databases, positions correspond to the "shape" of relations, and each position contains a value. Likewise, for XML, positions will correspond to the shape (that is more complex, since documents are modeled as trees), and they must have values associated with them. Consequently, we formally define the set of positions $Pos(T)$ in a tree $T = (V, lab, ele, att, root)$ as $\{(x, @l) \mid x \in V,\ att(x, @l) \text{ is defined}\}$. As before, we assume that there is an enumeration of positions (a bijection between $Pos(T)$ and $\{1, \ldots, n\}$ where $n = |Pos(T)|$) and we shall associate positions with their numbers in the enumeration. We define $adom(T)$ as the set of all values of attributes in T and $T_{p \leftarrow a}$ as an XML tree constructed from T by replacing the value in position p by a.

As in the relational case, we take the domain of values V of the attributes to be $\mathbb{N}^+$. Let Σ be a set of FDs over a DTD D and $k > 0$. Define $inst(D, \Sigma)$ as the set of all XML trees that conform to D and satisfy Σ and $inst_k(D, \Sigma)$ as its restriction to trees T with $adom(T) \subseteq [1, k]$. Now fix $T \in inst_k(D, \Sigma)$ and $p \in Pos(T)$. With the above definitions, we define the probability spaces $\mathcal{A}(T, p)$ and $\mathcal{B}^k_\Sigma(T, p)$ exactly as we defined $\mathcal{A}(I, p)$ and $\mathcal{B}^k_\Sigma(I, p)$ for a relational instance I. That is, $\Omega(T, p)$ is the set of all tuples $\bar{a}$ of the form $(a_1, \ldots, a_{p-1}, a_{p+1}, \ldots, a_n)$ such that every a_i is either a variable, or the value T has in the corresponding position, $SAT^k_\Sigma(T_{(a,\bar{a})})$ as the set of all possible ways to assign values from $[1, k]$ to variables in $\bar{a}$ that result in a tree satisfying Σ, and the rest of the definition repeats the relational case one verbatim, substituting T for I.

We use the above definitions to define $\textsc{Inf}^k_T(p \mid \Sigma)$ as the entropy of $\mathcal{B}^k_\Sigma(T, p)$ given $\mathcal{A}(T, p)$:

$$\textsc{Inf}^k_T(p \mid \Sigma) \ \stackrel{\text{def}}{=}\ H(\mathcal{B}^k_\Sigma(T, p) \mid \mathcal{A}(T, p))\ .$$

As in the relational case, we can show that the limit

$$\lim_{k \to \infty} \frac{\textsc{Inf}^k_T(p \mid \Sigma)}{\log k}$$

exists, and we denote it by $\textsc{Inf}_T(p \mid \Sigma)$. Following the relational case, we introduce

Definition 7.4.1 *An XML specification (D, Σ) is* well-designed *if for every $T \in inst(D, \Sigma)$ and every $p \in Pos(T)$, $\textsc{Inf}_T(p \mid \Sigma) = 1$.*

Note that the information-theoretic definition of well-designed schema presented in Section 3.4 for relational data proved to be extremely robust, as it extended straightforwardly to a different data model: we only needed a new definition of $Pos(T)$ to use in place of $Pos(I)$, and $Pos(T)$ is simply an enumeration of all the places in a document where attribute values occur. As in the relational case, it is possible to show that well-designed XML and XNF coincide. Furthermore, it is also possible to establish a useful structural criterion for $\text{INF}_T(p \mid \Sigma) = 1$, namely that an XML specification (D, Σ) is well-designed if and only if one position of an arbitrary $T \in inst(D, \Sigma)$ can always be assigned a fresh value.

Theorem 7.4.2 *Let D be a DTD and Σ a set of FDs over D. Then the following are equivalent.*

1) (D, Σ) is well-designed.

2) (D, Σ) is in XNF.

3) For every $T \in inst(D, \Sigma)$, $p \in Pos(T)$ and $a \in \mathbb{N}^+ - adom(T)$, $T_{p \leftarrow a} \models \Sigma$.

The proof of the theorem follows rather closely the proof of Proposition 3.4.9, by replacing relational concepts by their XML counterparts.

Proof of Theorem 7.4.2: We will prove the chain of implications 1) $\Rightarrow$ 2) $\Rightarrow$ 3) $\Rightarrow$ 1).

1) $\Rightarrow$ 2) Assume that (D, Σ) is not in XNF. We will show that there exists $T \in inst(D, \Sigma)$ and $p \in Pos(T)$ such that $\text{INF}_T(p \mid \Sigma) < 1$.

Given that (D, Σ) is not in XNF, there exists a nontrivial FD $X \to q.@l \in (D, \Sigma)^+$ such that $X \to q \notin (D, \Sigma)^+$. Thus, there is $T \in inst(D, \Sigma)$ containing tree tuples t_1, t_2 such that $t_1(q') = t_2(q')$ and $t_1(q') \neq \perp$, for every $q' \in X$, and $t_1(q) \neq t_2(q)$. We may assume that $t_1(q) \neq \perp$ and $t_2(q) \neq \perp$ (if $t_1(q) = \perp$ or $t_2(q) = \perp$, then $t_1(q.@l) \neq t_2(q.@l)$, which would contradict $T \models \Sigma$). Let $x = t_1(q)$, p be the position of $(x, @l)$ in T and $a = t_1(q.@l)$. Let $\bar{a}_0$ be the vector in $\Omega(T, p)$ containing no variables. Given that $t_1(q) \neq t_2(q)$ and none of these values is $\perp$, for every $b \in [1, k] - \{a\}$, $T_{(b, \bar{a}_0)} \not\models \Sigma$. Thus, for every $b \in [1, k] - \{a\}$, $P(b \mid \bar{a}_0) = 0$. Now a straightforward application of Lemma 3.4.10 implies

$$\text{INF}_T(p \mid \Sigma) \;=\; \lim_{k \to \infty} \text{INF}^k_T(p \mid \Sigma) / \log k \;<\; 1.$$

This concludes the proof.

2) $\Rightarrow$ 3) Let (D, Σ) be an XML specification in XNF, $T \in \mathit{inst}(D, \Sigma)$, $p \in \mathit{Pos}(T)$ and $a \in \mathbb{N}^+ - \mathit{adom}(T)$. We prove that $T_{p \leftarrow a} \models \Sigma$.

Assume, to the contrary, that $T_{p \leftarrow a} \not\models \Sigma$. Then there exists a FD $X \to q \in \Sigma$ such that $T_{p \leftarrow a} \not\models X \to q$. Thus, there exists $t'_1, t'_2 \in \mathit{tuples}_D(T_{p \leftarrow a})$ such that $t'_1(q') = t'_2(q')$ and $t'_1(q') \neq \perp$, for every $q' \in X$, and $t'_1(q) \neq t'_2(q)$. Assume that these tuples were generated from tuples $t_1, t_2 \in \mathit{tuples}_D(T)$. Given that $a \in \mathbb{N}^+ - \mathit{adom}(T)$, $t_1(q') = t_2(q')$ and $t_1(q') \neq \perp$, for every $q' \in X$, and, therefore, $t_1(q) = t_2(q)$, since $T \models \Sigma$. If q is an element path, then $t'_1(q) = t_1(q)$ and $t'_2(q) = t_2(q)$, since $T_{p \leftarrow a}$ is constructed from T by modifying only the values of attributes. Thus, $t'_1(q) = t'_2(q)$, a contradiction. Assume that q is an attribute path of the form $q_1.@l$. In this case, $X \to q_1.@l$ is a nontrivial FD in Σ and, therefore, $X \to q_1 \in (D, \Sigma)^+$, since (D, Σ) is in XNF. We conclude that $t_1(q_1) = t_2(q_1)$. Given that q_1 is an element path, as in the previous case we conclude that $t'_1(q_1) = t'_2(q_1)$. Hence, $t'_1(q_1.@l) = t'_2(q_1.@l)$, again a contradiction.

3) $\Rightarrow$ 1) Let $T \in \mathit{inst}(D, \Sigma)$ and $p \in \mathit{Pos}(T)$. We have to prove that $\textrm{INF}_T(p \mid \Sigma) = 1$. To show this, it suffices to prove that

$$\lim_{k \to \infty} \frac{\textrm{INF}^k_T(p \mid \Sigma)}{\log k} \;\; \geq \;\; 1. \tag{7.1}$$

Let $n = |\mathit{Pos}(T)|$ and $k > 2n$ such that $T \in \mathit{inst}_k(D, \Sigma)$. If $\bar{a} \in \Omega(T, p)$ and $\mathit{var}(\bar{a})$ is the set of variables mentioned in $\bar{a}$, then for every $a \in [1, k] - \mathit{adom}(T)$,

$$|\mathit{SAT}^k_\Sigma(T_{(a,\bar{a})})| \;\; \geq \;\; (k - 2n)^{|\mathit{var}(\bar{a})|}$$

since by hypothesis one can replace values in positions of $\bar{a}$ one by one, provided that each position gets a fresh value. Thus, given that $\sum_{b \in [1,k]} |\mathit{SAT}^k_\Sigma(T_{(b,\bar{a})})| \leq k^{|\mathit{var}(\bar{a})|+1}$, for every $a \in [1, k] - \mathit{adom}(T)$ and every $\bar{a} \in \Omega(T, p)$, we have:

$$P(a \mid \bar{a}) \;\; \geq \;\; \frac{(k - 2n)^{|\mathit{var}(\bar{a})|}}{k^{|\mathit{var}(\bar{a})|+1}} \;\; = \;\; \frac{1}{k}\,(1 - \frac{2n}{k})^{|\mathit{var}(\bar{a})|}. \tag{7.2}$$

Functional dependencies are generic constraints. Thus, for every $a, b \in [1, k] - \mathit{adom}(T)$ and every $\bar{a} \in \Omega(T, p)$, $P(a \mid \bar{a}) = P(b \mid \bar{a})$. Hence, for every $a \in [1, k] - \mathit{adom}(T)$ and every $\bar{a} \in \Omega(T, p)$:

$$P(a \mid \bar{a}) \;\; \leq \;\; \frac{1}{k - |\mathit{adom}(T)|} \;\; \leq \;\; \frac{1}{k - n}. \tag{7.3}$$

In order to prove (7.1), we need to establish a lower bound for $\textrm{INF}^k_T(p \mid \Sigma)$. We do this by using (7.2) and (7.3) as follows: Given the term $P(a \mid \bar{a}) \log \frac{1}{P(a|\bar{a})}$, we use (7.2) and

(7.3) to replace $P(a \mid \bar{a})$ and $\log \frac{1}{P(a|\bar{a})}$ by smaller terms, respectively. More precisely,

$$
\begin{aligned}
\textsc{Inf}_T^k(p \mid \Sigma) &= \sum_{\bar{a} \in \Omega(T,p)} \left(P(\bar{a}) \sum_{a \in [1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \right) \\
&\geq \frac{1}{2^{n-1}} \sum_{a \in [1,k]-adom(T)} \sum_{\bar{a} \in \Omega(T,p)} \frac{1}{k} \, (1 - \frac{2n}{k})^{|var(\bar{a})|} \log(k-n) \\
&= \frac{1}{2^{n-1}} \, \log(k-n) \, \frac{1}{k} \sum_{a \in [1,k]-adom(I)} \sum_{i=0}^{n-1} \binom{n-1}{i} (1 - \frac{2n}{k})^i \\
&= \frac{1}{2^{n-1}} \, \log(k-n) \, \frac{1}{k} \sum_{a \in [1,k]-adom(I)} ((1 - \frac{2n}{k}) + 1)^{n-1} \\
&\geq \frac{1}{2^{n-1}} \, \log(k-n) \, \frac{1}{k} \, (k-n) \, (2 - \frac{2n}{k})^{n-1} \\
&= \frac{1}{2^{n-1}} \, \log(k-n) \, (1 - \frac{n}{k}) \, 2^{n-1} \, (1 - \frac{n}{k})^{n-1} \\
&= \log(k-n) \, (1 - \frac{n}{k})^n.
\end{aligned}
$$

Therefore, $\frac{\textsc{Inf}_T^k(p|\Sigma)}{\log k} \geq \frac{\log(k-n)}{\log k}(1 - \frac{n}{k})^n$. Since $\lim_{k \to \infty} \frac{\log(k-n)}{\log k}(1 - \frac{n}{k})^n = 1$, (7.1) follows. This concludes the proof. □

The theory of XML constraints and normal forms is not nearly as advanced as its relational counterparts, but we have demonstrated here that the definition of well-designed schemas works well for the existing normal form based on FDs; thus, it could be used to test other design criteria for XML when they are proposed.

7.5 Normalization Algorithms

The goal of this section is to show how to transform a DTD D and a set of FDs Σ into a new specification (D', Σ') that is in XNF and contains the same information.

Throughout the section, we assume that the DTDs are non-recursive. This can be done without any loss of generality. Notice that in a recursive DTD D, the set of all paths is infinite. However, a given set of FDs Σ only mentions a finite number of paths, which means that it suffices to restrict one's attention to a finite number of "unfoldings" of recursive rules.

We make an additional assumption that all the FDs are of the form: $\{q, p_1.@l_1, \ldots, p_n.@l_n\} \to p$. That is, they contain at most one element path on the left-hand side. Note that all the FDs we have seen so far are of this form. While constraints of the form $\{q, q', \ldots\}$ are not forbidden, they appear to be quite unnatural (in fact it is very hard to come up with a reasonable example where they could be used). Furthermore, even if we have such constraints, they can be easily eliminated. To do so, we create a new attribute $@l$, remove $\{q, q'\} \cup S \to p$ and replace it by $q'.@l \to q'$ and $\{q, q'.@l\} \cup S \to p$.

We shall also assume that paths do not contain the symbol $\texttt{S}$ (since $p.\texttt{S}$ can always be replaced by a path of the form $p.@l$).

7.5.1 The Decomposition Algorithm

In this section, we present an algorithm for converting an XML schema into a new schema in XNF. This algorithm combines two basic ideas presented in the introduction of this chapter: creating a new element type, and moving an attribute. It should be noted that the former step resembles the decomposition step of BCNF normalization algorithms (see Section 2.1.3). The more we apply this step, the less expensive it is to update XML documents since they contain less redundancy, and the more expensive it is to query them since the original document has to be recomposed. Depending on how important are these operations, the user can choose not to apply the algorithm until a schema in XNF is obtained; instead he/she can apply a few steps of the algorithm or even use a completely unnormalized XML schema. In general, the right amount of normalization, both in the relational and in the XML cases, should depend on a query workload. The problem of finding this amount is an optimization problem that should take into account the cost of reconstructing the original database. In relational databases, this cost is associated with the cost of performing joins between different tables, which is a well studied problem. In the case of XML, how to measure this cost is not so clear as it depends on performing queries in an XML query language like XQuery [BCF$^+$], which is under development and does not have a well studied cost model. To the best of our knowledge, the problem of finding the right amount of normalization has received very little attention in the database community, and there are only a few papers on the subject [SS82]. This problem and, in particular, the problem of measuring the cost of reconstructing XML documents are out of the scope of this dissertation, and they are interesting problems for future research.

It should also be noted that since decomposition and recomposition are in general expensive operations, the normalization algorithm presented in this section also includes a simpler step that moves attributes in XML specifications. This step takes into account the hierarchical structure of XML documents, and it does not decompose the original schema. The decomposition step is only applied when this step cannot be applied. We have found real applications where this simple step can be used to solve some normalization problems (e.g. DBLP [Ley]) and, thus, we expect it to be used in practice to avoid expensive recompositions.

For presenting the algorithm and proving its losslessness, we make the following assumption: if $X \to p.@l$ is an FD that causes a violation of XNF, then every time that $p.@l$ is not null, every path in X is not null. This will make our presentation simpler, and then at the end of the section we will show how to eliminate this assumption.

Given a DTD D and a set of FDs Σ, a nontrivial FD $S \to p.@l$ is called *anomalous*, over (D, Σ), if it violates XNF; that is, $S \to p.@l \in (D, \Sigma)^+$ but $S \to p \notin (D, \Sigma)^+$. A path on the right-hand side of an anomalous FD is called an anomalous path, and the set of all such paths is denoted by $AP(D, \Sigma)$.

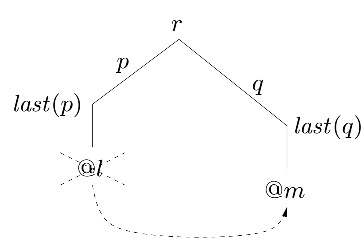
Next we present the two steps of the XNF decomposition algorithm: creating a new element type, and moving an attribute.

Moving attributes

Let $D = (E,\ A,\ P,\ R,\ r)$ be a DTD and Σ a set of FDs over D. Assume that (D, Σ) contains an anomalous FD $q \to p.@l$, where $q \in \mathit{EPaths}(D)$. For example, the DBLP database shown in example 7.1.2 contains an anomalous FD of this form:

$$db.conf.issue \to db.conf.issue.inproceedings.@year. \tag{7.4}$$

To eliminate the anomalous FD, we move the attribute $@l$ from the set of attributes of the last element of p to the set of attributes of the last element of q, as shown in the following figure



For instance, to eliminate the anomalous functional dependency (7.4) we move the attribute *@year* from the set of attributes of *inproceedings* to the set of attributes of *issue*.

Formally, the new DTD $D[p.@l := q.@m]$, where $@m$ is an attribute, is defined to be (E, A', P, R', r), where $A' = A \cup \{@m\}$, $R'(last(q)) = R(last(q)) \cup \{@m\}$, $R'(last(p)) = R(last(p)) - \{@l\}$ and $R'(\tau') = R(\tau')$ for each $\tau' \in E - \{last(q), last(p)\}$.

After transforming D into a new DTD $D[p.@l := q.@m]$, a new set of functional dependencies is generated. Formally, the set of FDs $\Sigma[p.@l := q.@m]$ over $D[p.@l := q.@m]$ consists of all FDs $S_1 \to S_2 \in (D, \Sigma)^+$ with $S_1 \cup S_2 \subseteq paths(D[p.@l := q.@m])$. Observe that the new set of FDs does not include the functional dependency $q \to p.@l$ and, thus, it contains a smaller number of anomalous paths, as we show in the following proposition.

Proposition 7.5.1 *Let D be a DTD, Σ a set of FDs over D, $q \to p.@l$ an anomalous FD, with $q \in EPaths(D)$, $D' = D[p.@l := q.@m]$, where $@m$ is not an attribute of $last(q)$, and $\Sigma' = \Sigma[p.@l := q.@m]$. Then $AP(D', \Sigma') \subsetneq AP(D, \Sigma)$.*

PROOF: First, we prove (by contradiction) that $q.@m \notin AP(D', \Sigma')$. Suppose that $S' \subseteq paths(D')$ and $S' \to q.@m \in (D', \Sigma')^+$ is a nontrivial functional dependency. Assume that $S' \to q \notin (D', \Sigma')^+$. Then there is an XML tree T' such that $T' \models (D', \Sigma')$ and T' contains tree tuples t_1, t_2 such that $t_1.S' = t_2.S'$, $t_1.S' \neq \perp$ and $t_1.q \neq t_2.q$. Given that there is no a constraint in Σ' including the path $q.@m$, the XML tree T'' constructed from T' by giving two distinct values to $t_1.q.@m$ and $t_2.q.@m$ conforms to D', satisfies Σ' and does not satisfy $S' \to q.@m$, a contradiction. Hence, $q.@m \notin AP(D', \Sigma')$.

Second, we prove that for every $S_1 \cup S_2 \subseteq paths(D') - \{q.@m\}$, $(D, \Sigma) \vdash S_1 \to S_2$ if and only if $(D', \Sigma') \vdash S_1 \to S_2$, and, thus, by considering the previous paragraph we conclude that $AP(D', \Sigma') \subseteq AP(D, \Sigma)$. Let $S_1 \cup S_2 \subseteq paths(D') - \{q.@m\}$. By definition of Σ', we know that if $(D, \Sigma) \vdash S_1 \to S_2$, then $(D', \Sigma') \vdash S_1 \to S_2$ and, therefore, we only need to prove the other direction. Assume that $(D, \Sigma) \nvdash S_1 \to S_2$. Then there exists an XML tree T such that $T \models (D, \Sigma)$ and $T \not\models S_1 \to S_2$. Define an XML tree T' from T by assigning arbitrary values to $q.@m$ and removing the attribute $@l$ from $last(p)$. Then $T' \models (D', \Sigma')$ and $T' \not\models S_1 \to S_2$, since all the paths mentioned in $\Sigma' \cup \{S_1 \to S_2\}$ are included in $paths(D') - \{q.@m\}$. Thus, $(D', \Sigma') \nvdash S_1 \to S_2$.

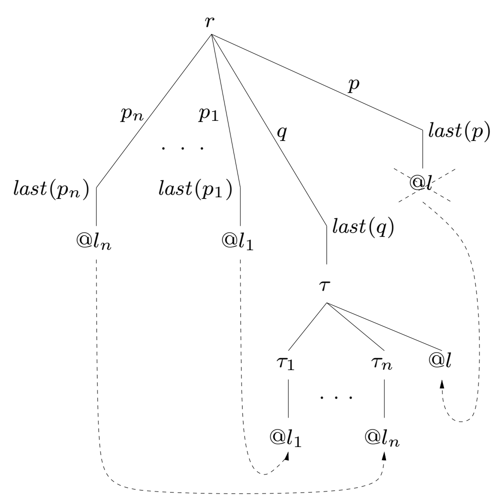
To conclude the proof we note that $p.@l \in AP(D, \Sigma)$ and $p.@l \notin AP(D', \Sigma')$, since $p.@l \notin paths(D')$. Therefore, $AP(D', \Sigma') \subsetneq AP(D, \Sigma)$. □

Creating new element types

Let $D = (E,\ A,\ P,\ R,\ r)$ be a DTD and Σ a set of FDs over D. Assume that (D, Σ) contains an anomalous FD $\{q, p_1.@l_1, \ldots, p_n.@l_n\} \rightarrow p.@l$, where $q \in \mathit{EPaths}(D)$ and $n \geq 1$. For example, the university database shown in example 7.1.1 contains an anomalous FD of this form (considering *name*.S as an attribute of *student*):

$$\{\mathit{courses},\ \mathit{courses.course.taken\_by.student.@sno}\} \rightarrow \\ \mathit{courses.course.taken\_by.student.name}.\mathtt{S}. \quad (7.5)$$

To eliminate the anomalous FD, we create a new element type τ as a child of the last element of q, we make $\tau_1, \ldots, \tau_n$ its children, where $\tau_1, \ldots, \tau_n$ are new element types, we remove $@l$ from the list of attributes of $\mathit{last}(p)$ and we make it an attribute of τ and we make $@l_1, \ldots, @l_n$ attributes of $\tau_1, \ldots, \tau_n$, respectively, but without removing them from the sets of attributes of $\mathit{last}(p_1), \ldots, \mathit{last}(p_n)$, as shown in the following figure.



For instance, to eliminate the anomalous functional dependency (7.5), in example 7.1.1 we create a new element type *info* as a child of *courses*, we remove *name*.S from *student* and we make it an "attribute" of *info*, we create an element type *number* as a child of *info* and we make *@sno* its attribute. We note that we do not remove *@sno* as an attribute of *student*. Formally, if $\tau, \tau_1, \ldots, \tau_n$ are element types which are not in E, the new DTD, denoted by $D[p.@l := q.\tau[\tau_1.@l_1, \ldots, \tau_n.@l_n, @l]]$, is (E', A, P', R', r), where $E' = E \cup \{\tau, \tau_1, \ldots, \tau_n\}$ and

1. if $P(\mathit{last}(q))$ is a regular expression s, then $P'(\mathit{last}(q))$ is defined as the concatenation of s and τ^*, that is (s, τ^*). Furthermore, $P'(\tau)$ is defined as the concate-

nation of τ_1^*, ..., τ_n^*, $P'(\tau_i) = \epsilon$, for each $i \in [1,n]$, and $P'(\tau') = P(\tau')$, for each $\tau' \in E - \{last(q)\}$.

2. $R'(\tau) = \{@l\}$, $R'(\tau_i) = \{@l_i\}$, for each $i \in [1,n]$, $R'(last(p)) = R(last(p)) - \{@l\}$ and $R'(\tau') = R(\tau')$ for each $\tau' \in E - \{last(p)\}$.

After transforming D into a new DTD $D' = D[p.@l := q.\tau[\tau_1.@l_1, \ldots, \tau_n.@l_n, @l]]$, a new set of functional dependencies is generated. Formally, $\Sigma[p.@l := q.\tau[\tau_1.@l_1, \ldots, \tau_n.@l_n, @l]]$ is a set of FDs over D' defined as the union of the sets of constraints defined in 1., 2. and 3.:

1. $S_1 \to S_2 \in (D, \Sigma)^+$ with $S_1 \cup S_2 \subseteq paths(D')$.

2. Each FD over q, p_i, $p_i.@l_i$ ($i \in [1,n]$) and $p.@l$ is transferred to τ and its children. That is, if $S_1 \cup S_2 \subseteq \{q, p_1, \ldots, p_n, p_1.@l_1, \ldots, p_n.@l_n, p.@l\}$ and $S_1 \to S_2 \in (D, \Sigma)^+$, then we include an FD obtained from $S_1 \to S_2$ by changing p_i to $q.\tau.\tau_i$, $p_i.@l_i$ to $q.\tau.\tau_i.@l_i$, and $p.@l$ to $q.\tau.@l$.

3. $\{q,\ q.\tau.\tau_1.@l_1,\ \ldots,\ q.\tau.\tau_n.@l_n\} \to\ q.\tau$, and $\{q.\tau,\ q.\tau.\tau_i.@l_i\} \to q.\tau.\tau_i$ for $i \in [1,n]$ [1].

We are not interested in applying this transformation to an arbitrary anomalous FD, but rather to a *minimal* one. To understand the notion of minimality for XML FDs, we first introduce this notion for relational databases. Let R be a relation schema containing a set of attributes U and Σ be a set of FDs over R. If (R, Σ) is not in BCNF, then there exist pairwise disjoint sets of attributes X, Y and Z such that $U = X \cup Y \cup Z$, $\Sigma \vdash X \to Y$ and $\Sigma \not\vdash X \to A$, for every $A \in Z$. In this case we say that $X \to Y$ is an anomalous FD. To eliminate this anomaly, a decomposition algorithm splits relation R into two relations: $S(X,Y)$ and $T(X,Z)$. A desirable property of the new schema is that S or T is in BCNF. We say that $X \to Y$ is a minimal anomalous FD if $S(X,Y)$ is in BCNF, that is, $S(X,Y)$ does not contain an anomalous FD. This condition can be defined as follows: $X \to Y$ is *minimal* if there are no pairwise disjoint sets $X', Y' \subseteq U$ such that $X' \cup Y' \subsetneq X \cup Y$, $\Sigma \vdash X' \to Y'$ and $\Sigma \not\vdash X' \to X \cup Y$.

In the XML context, the definition of minimality is similar in the sense that we expect the new element types τ, τ_1, ..., τ_n form a structure not containing anomalous

[1] If $\perp$ can be a value of $p.@l$ in $tuples_D(T)$, the definition must be modified slightly, by letting $P'(\tau)$ be $\tau_1^*, \ldots, \tau_n^*, (\tau'|\epsilon)$, where τ' is fresh, making $@l$ an attribute of τ', and modifying the definition of FDs accordingly.

elements. However, the definition of minimality is more complex to account for paths used in FDs. We say that $\{q, p_1.@l_1, \ldots, p_n.@l_n\} \rightarrow p_0.@l_0$ is (D,Σ)-minimal if there is no anomalous FD $S' \rightarrow p_i.@l_i \in (D,\Sigma)^+$ such that $i \in [0,n]$ and S' is a subset of $\{q, p_1, \ldots, p_n, p_0.@l_0, \ldots, p_n.@l_n\}$ such that $|S'| \leq n$ and S' contains at most one element path.

Proposition 7.5.2 *Let D be a DTD, Σ a set of FDs over D and $\{q, p_1.@l_1, \ldots, p_n.@l_n\} \rightarrow p.@l$ a (D,Σ)-minimal anomalous FD, where $q \in EPaths(D)$ and $n \geq 1$. If $D' = D[p.@l := q.\tau[\tau_1.@l_1, \ldots, \tau_n.@l_n, @l]]$, where τ, τ_1, $\ldots$, τ_n are new element types, and $\Sigma' = \Sigma[p.@l := q.\tau[\tau_1.@l_1, \ldots, \tau_n.@l_n, @l]]$, then $AP(D', \Sigma') \subsetneq AP(D,\Sigma)$.*

Proof: First, we prove that $q.\tau.\tau_i.@l_i \notin AP(D',\Sigma')$, for each $i \in [1,n]$. Suppose that there is $S' \subseteq paths(D')$ such that $S' \rightarrow q.\tau.\tau_i.@l_i$ is a nontrivial functional dependency in $(D',\Sigma')^+$ for some $i \in [1,n]$. Notice that $q.\tau.\tau_i \notin S'$, since $q.\tau.\tau_i \rightarrow q.\tau.\tau_i.@l_i$ is a trivial functional dependency. Let $S_1 \cup S_2 = S'$, where (1) $S_1 \cap (\{q, q.\tau.@l\} \cup \{q.\tau.\tau_j \mid j \in [1,n]$ and $j \neq i\} \cup \{q.\tau.\tau_j.@l_j \mid j \in [1,n]\}) = \emptyset$ and (2) $S_2 \subseteq \{q, q.\tau.@l\} \cup \{q.\tau.\tau_j \mid j \in [1,n]$ and $j \neq i\} \cup \{q.\tau.\tau_j.@l_j \mid j \in [1,n]\}$.

If there is no an XML tree T' conforming to D', satisfying Σ' and containing a tuple t such that $t.S_1 \cup S_2 \neq \perp$, then $S_1 \cup S_2 \rightarrow q.\tau.\tau_i$ must be in $(D',\Sigma')^+$. In this case $q.\tau.\tau_i.@l_i \notin AP(D',\Sigma')$. Suppose that there is an XML tree T' conforming to D', satisfying Σ' and containing a tuple t such that $t.S_1 \cup S_2 \neq \perp$. In this case, by definition of Σ' it is straightforward to prove that $S_2 \rightarrow q.\tau.\tau_i.@l_i$ is in $(D',\Sigma')^+$.

By definition of Σ' and (D,Σ)-minimality of $\{q, p_1.@l_1, \ldots, p_n.@l_n\} \rightarrow p.@l$, one of the following is true: (1) $S_2 \rightarrow q.\tau.\tau_i.@l_i$ is not an anomalous FD, (2) $\{q,\ q.\tau.\tau_1.@l_1,\ \ldots,\ q.\tau.\tau_n.@l_n,\ q.\tau.@l\} = S_2 \cup \{q.\tau.\tau_i.@l_i\}$ or (3) $\{q.\tau.\tau_j, q.\tau.\tau_1.@l_1,\ \ldots,\ q.\tau.\tau_n.@l_n,\ q.\tau.@l\} = S_2 \cup \{q.\tau.\tau_i.@l_i\}$ for some $j \neq i$ ($j \in [1,n]$). In the first case, $q.\tau.\tau_i.@l_i \notin AP(D',\Sigma')$, so we assume that either (2) or (3) holds. We prove that $S_2 \rightarrow q.\tau.\tau_i$ must be in $(D',\Sigma')^+$. If either (2) or (3) holds, then $S_2 \cup \{q.\tau.\tau_i.@l_i\} \rightarrow q.\tau$ is in $(D',\Sigma')^+$ since $\{q,\ q.\tau.\tau_1.@l_1,\ \ldots,\ q.\tau.\tau_n.@l_n\} \rightarrow q.\tau$ is in Σ' and $q.\tau.\tau_k \rightarrow q$ is a trivial FD in D', for every $k \in [1,n]$. Let T' be an XML tree conforming to D' and satisfying Σ' and $t_1, t_2 \in tuples_{D'}(T')$ such that $t_1.S_2 = t_2.S_2$ and $t_1.S_2 \neq \perp$. Given that $S_2 \rightarrow q.\tau.\tau_i.@l_i \in (D',\Sigma')^+$, $t_1.q.\tau.\tau_i.@l_i = t_2.q.\tau.\tau_i.@l_i$. If $t_1.q.\tau.\tau_i.@l_i = \perp$, then $t_1.q.\tau.\tau_i = t_2.q.\tau.\tau_i = \perp$. If $t_1.q.\tau.\tau_i.@l_i \neq \perp$, then $t_1.q.\tau = t_2.q.\tau$ and $t_1.q.\tau \neq \perp$, because $S_2 \cup \{q.\tau.\tau_i.@l_i\} \rightarrow q.\tau \in (D',\Sigma')^+$. But, by definition of Σ', $\{q.\tau,\ q.\tau.\tau_i.@l_i\} \rightarrow q.\tau.\tau_i \in \Sigma'$, and, therefore, $t_1.q.\tau.\tau_i = t_2.q.\tau.\tau_i$. In any case, we conclude that $t_1.q.\tau.\tau_i = t_2.q.\tau.\tau_i$ and, therefore, $S_2 \to q.\tau.\tau_i \in (D', \Sigma')^+$. Thus, $q.\tau.\tau_i.@l_i \notin AP(D', \Sigma')$.

In a similar way, we conclude that $q.\tau.@l \notin AP(D', \Sigma')$.

Second, we prove that for every $S_3 \cup S_4 \subseteq \mathit{paths}(D) - \{p.@l\}$, $(D, \Sigma) \vdash S_3 \to S_4$ if and only if $(D', \Sigma') \vdash S_3 \to S_4$, and, thus, by considering the previous paragraph we conclude that $AP(D', \Sigma') \subseteq AP(D, \Sigma)$. Let $S_3 \cup S_4 \subseteq \mathit{paths}(D) - \{p.@l\}$. By definition of Σ', we know that if $(D, \Sigma) \vdash S_3 \to S_4$, then $(D', \Sigma') \vdash S_3 \to S_4$ and, therefore, we only need to prove the other direction. Assume that $(D, \Sigma) \nvdash S_3 \to S_4$. Then there exists an XML tree T such that $T \models (D, \Sigma)$ and $T \not\models S_3 \to S_4$. Define an XML tree T' from T by assigning $\perp$ to $q.\tau$ and removing the attribute $@l$ from $\mathit{last}(p)$. Then $T' \models (D', \Sigma')$ and $T' \not\models S_3 \to S_4$, since all the paths mentioned in $\Sigma' \cup \{S_3 \to S_4\}$ are included in $\mathit{paths}(D) - \{p.@l\}$. Thus, $(D', \Sigma') \nvdash S_3 \to S_4$.

To conclude the proof we note that $p.@l \in AP(D, \Sigma)$ and $p.@l \notin AP(D', \Sigma')$, since $p.@l \notin \mathit{paths}(D')$. Therefore, $AP(D', \Sigma') \subsetneq AP(D, \Sigma)$. $\square$

The algorithm

The algorithm applies the two transformations presented in the previous sections until the schema is in XNF, as shown in Figure 7.4. Step (2) of the algorithm corresponds to the "moving attributes" rule applied to an anomalous FD $q \to p.@l$ and step (3) corresponds to the "creating new element types" rule applied to an anomalous FD $\{q, p_1.@l_1, \ldots, p_n.@l_n\} \to p.@l$. We choose to apply first the "moving attributes" rule since the other one involves minimality testing .

The algorithm shows in Figure 7.4 involves FD implication, that is, testing membership in $(D, \Sigma)^+$ (and consequently testing XNF and (D, Σ)-minimality), which is described in Section 6.3. Since each step reduces the number of anomalous paths (Propositions 7.5.1 and 7.5.2), we obtain:

Theorem 7.5.3 *The XNF decomposition algorithm terminates, and outputs a specification (D, Σ) in XNF.*

Even if testing FD implication is infeasible, one can still decompose into XNF, although the final result may not be as good as with using the implication. A slight modification of the proof of Propositions 7.5.1 and 7.5.2 yields:

(1) If (D, Σ) is in XNF then return (D, Σ), otherwise go to step (2).

(2) If there is an anomalous FD $X \to p.@l$ and $q \in \mathit{EPaths}(D)$ such that $q \in X$ and $q \to X \in (D, \Sigma)^+$, then:

  (2.1) Choose a fresh attribute $@m$

  (2.2) $D := D[p.@l := q.@m]$

  (2.3) $\Sigma := \Sigma[p.@l := q.@m]$

  (2.4) Go to step (1)

(3) Choose a (D, Σ)-minimal anomalous FD $X \to p.@l$,
where $X = \{q,\, p_1.@l_1,\, \ldots,\, p_n.@l_n\}$

  (3.1) Create fresh element types $\tau,\, \tau_1,\, \ldots,\, \tau_n$

  (3.2) $D := D[p.@l := q.\tau[\tau_1.@l_1,\, \ldots,\, \tau_n.@l_n,\, @l]]$

  (3.3) $\Sigma := \Sigma[p.@l := q.\tau[\tau_1.@l_1,\, \ldots,\, \tau_n.@l_n,\, @l]]$

  (3.4) Go to step (1)

Figure 7.4: XNF decomposition algorithm.

Proposition 7.5.4 *Consider a simplification of the XNF decomposition algorithm which only consists of step (3) applied to FDs $S \to p.@l \in \Sigma$, and in which the definition of $\Sigma[p.@l := q.\tau[\tau_1.@l_1,\, \ldots,\, \tau_n.@l_n,\, @l]]$ is modified by using Σ instead of $(D, \Sigma)^+$. Then such an algorithm always terminates and its result is in XNF.*

7.5.2 Lossless Decomposition

To prove that our transformations do not lose any information from the documents, we define the concept of lossless decompositions similarly to the relational notion of "calculously dominance" from [Hul86]. That notion requires the existence of two relational algebra queries that translate back and forth between two relational schemas. Adapting the definition of [Hul86] is problematic in our setting, as no XML query language yet has the same "yardstick" status as relational algebra for relational databases.

Instead, we define (D', Σ') as a lossless decomposition of (D, Σ) if there is a mapping f from paths in the DTD D' to paths in the DTD D such that for every tree $T \models (D, \Sigma)$, there is a tree $T' \models (D', \Sigma')$ such that T and T' agree on all the paths with respect to this mapping f.

This can be done formally using the relational representation of XML trees via the $tuples_D(\cdot)$ operator. Given DTDs D and D', a function $f : paths(D') \to paths(D)$ is a *mapping from D' to D* if f is onto and a path p is an element path in D' if and only if $f(p)$ is an element path in D. Given tree tuples $t \in \mathcal{T}(D)$ and $t' \in \mathcal{T}(D')$, we write $t \equiv_f t'$ if for all $p \in paths(D') - EPaths(D')$, $t'.p = t.f(p)$. Given nonempty sets of tree tuples $X \subseteq \mathcal{T}(D)$ and $X' \subseteq \mathcal{T}(D')$, we let $X \equiv_f X'$ if for every $t \in X$, there exists $t' \in X'$ such that $t \equiv_f t'$, and for every $t' \in X'$, there exist $t \in X$ such that $t \equiv_f t'$. Finally, if T and T' are XML trees such that $T \lhd D$ and $T' \lhd D'$, we write $T \equiv_f T'$ if $tuples_D(T) \equiv_f tuples_{D'}(T')$.

Definition 7.5.5 *Given XML specifications (D, Σ) and (D', Σ'), (D', Σ') is a lossless decomposition of (D, Σ), written $(D, \Sigma) \leq_{\text{lossless}} (D', \Sigma')$, if there exists a mapping f from D' to D such that for every $T \models (D, \Sigma)$ there is $T' \models (D', \Sigma')$ such that $T \equiv_f T'$.*

In other words, all information about a document conforming to (D, Σ) can be recovered from some document that conforms to (D', Σ').

It follows immediately from the definition that $\leq_{\text{lossless}}$ is transitive. Furthermore, we show that every step of the normalization algorithm is lossless.

Proposition 7.5.6 *If (D', Σ') is obtained from (D, Σ) by using one of the transformations from the normalization algorithm, then $(D, \Sigma) \leq_{\text{lossless}} (D', \Sigma')$.*

Proof: We consider the two steps of the normalization algorithm, and for each step generate a mapping f. The proofs that those mappings satisfy the conditions of Definition 7.5.5 are straightforward.

1. Assume that the "moving attribute" transformation was used to generate (D', Σ'). Then $D' = D[p.@l := q.@m]$, $\Sigma' = \Sigma[p.@l := q.@m]$ and $q \to p.@l$ is an anomalous FD in $(D, \Sigma)^+$. In this case, the mapping f from D' to D is defined as follows. For every $p' \in paths(D') - \{q.@m\}$, $f(p') = p'$, and $f(q.@m) = p.@l$.

2. Assume that the "creating new element types" transformation was used to generate (D', Σ'). Then (D', Σ') was generated by considering a (D, Σ)-minimal anomalous FD $\{q,\ p_1.@l_1,\ \ldots,\ p_n.@l_n\} \to p.@l$. Thus, $D' = D[p.@l := q.\tau[\tau_1.@l_1, \ldots, \tau_n.@l_n, @l]]$ and $\Sigma' = \Sigma[p.@l := q.\tau[\tau_1.@l_1, \ldots, \tau_n.@l_n, @l]]$. In this case, the mapping f from D' to D is defined as follows: $f(q.\tau) = p$, $f(q.\tau.@l) = p.@l$, $f(q.\tau.\tau_i) = p_i$, $f(q.\tau.\tau_i.@l_i) = p_i.@l_i$ and $f(p') = p'$ for the remaining paths $p' \in paths(D')$.

□

Thus, if (D', Σ') is the output of the normalization algorithm on (D, Σ), then $(D, \Sigma) \leq_{\text{lossless}} (D', \Sigma')$.

In relational databases, the definition of lossless decomposition indicates how to transform instances containing redundant information into databases without redundancy. This transformation uses the projection operator. Notice that Definition 7.5.5 also indicates a way of transforming XML documents to generate well-designed documents: If $(D, \Sigma) \leq_{\text{lossless}} (D', \Sigma')$, then for every $T \models (D, \Sigma)$ there exists $T' \models (D', \Sigma')$ such that T and T' contain the same data values. The mappings $T \mapsto T'$ corresponding to the two transformations of the normalization algorithm can be implemented in an XML query language, more precisely, using XQuery FLWOR[2] expressions. We use transformations of documents shown in Section 7.1 for illustration; the reader will easily generalize them to produce the general queries corresponding to the transformations of the normalization algorithm.

Example 7.5.7 Assume that the DBLP database is stored in a file `dblp.xml`. As shown in example 7.1.2, this document can contain redundant information since year is stored multiple times for a given conference. We can solve this problem by applying the "moving attribute" transformation and making year an attribute of issue. This transformation can be implementing by using the following FLWOR expression:

```
let $root := document("dblp.xml")/db
<db>
{ for $co in $root/conf
  <conf>
    <title> { $co/title/text() } </title>,
    { for $is in $co/issue
      let $value := $is/inproceedings[position() = 1]/@year
      <issue year="{ $value }">
      { for $in in $is/inproceedings
        <inproceedings key="{ $in/@key }" pages="{ $in/@pages }">
        { for $au in $in/author
          <author> { $au/text() } </author>,
          <title> { $in/title/text() } </title>
```

[2]FLWOR stands for *for*, *let*, *where*, *order by*, and *return*.

```
        }
        </inproceedings>
      }
      </issue>
    }
  </conf>
}
</db>
```

The XPath expression `$is/inproceedings[position() = 1]/@year` is used to retrieve for every issue the value of the attribute year in the first paper in that issue. For every issue this number is stored in a variable `$value` and it becomes the value of its attribute year: `<issue year="{ $value }">`. □

Example 7.5.8 Assume that the XML document shown in Figure 7.1 is stored in a file `university.xml`. This document stores information about courses in a university and it contains redundant information since for every student taking a course we store his/her name. To solve this problem, we split the information about names and grades by creating an extra element type, `info`, for student information. This transformation can be implemented as follows.

```
let $root := document("university.xml")/courses
<courses>
{ for $co in $root/course
  <course> {-- Query that removes name as a child of student --} </course>,
  for $na in distinct-values($root/course/taken_by/student/name/text())
  <info>
  { for $nu in distinct-values($root/course/taken_by/student[name/text() =
                                                             $na]/@sno)
    <number sno="{ $nu }">,
    <name> { $na } </name>
  }
  </info>
}
</courses>
```

We omitted the query that removes `name` as a child of `student` since it can be done as in the previous example. □

7.5.3 Justifying the Decomposition Algorithm

We now show how the information-theoretic measure of Section 7.4 can be used for reasoning about normalization algorithms at the instance level. The results shown here state that after each step of the decomposition algorithm, the amount of information in each position does not decrease.

We shall prove a result similar to Theorem 3.5.1 of Section 3.5. To state the result, we need to explain how each step of the decomposition algorithm induces a mapping between positions in two XML trees. Recall that this algorithm eliminates anomalous functional dependencies by using two basic steps: moving an attribute, and creating a new element type.

Let (D, Σ) be an XML specification and $T \in \mathit{inst}(D, \Sigma)$. Assume that (D, Σ) is not in XNF. Let (D', Σ') be an XML specification obtained by executing one step of the normalization algorithm. Every step of this algorithm induces a natural transformation on XML documents. One of the properties of the algorithm is that for each normalization step that transforms $T \in \mathit{inst}(D, \Sigma)$ into $T' \in \mathit{inst}(D', \Sigma')$, one can find a map $\pi_{T',T} : \mathit{Pos}(T') \to 2^{\mathit{Pos}(T)}$ that associates each position in the new tree T' with one or more positions in the old tree T, as shown below.

1) Assume that $D' = D[q.@l := q'.@m]$ and, therefore, $q' \to q.@l$ is an anomalous FD in (D, Σ). In this case, an XML tree T' is constructed from T as follows. For every $t \in \mathit{tuples}_D(T)$, define a tree tuple t' by using the following rule: $t'(q'.@m) = t(q.@l)$ and for every $q'' \in \mathit{paths}(D) - \{q.@l\}$, $t'(q'') = t(q'')$. Then T' is an XML tree whose tree tuples are $\{t' \mid t \in \mathit{tuples}_D(T)\}$. Furthermore, positions in t' are associated to positions in t as follows: if $p' = (t'(q'), @m)$, then $\pi_{T',T}(p') = \{(t(q), @l)\}$; otherwise, $\pi_{T',T}(p') = \{p'\}$.

2) Assume that (D', Σ') was generated by considering a (D, Σ)-minimal anomalous FD $\{q', q_1.@l_1, \ldots, q_n.@l_n\} \to q.@l$. Thus, $D' = D[q.@l := q'.a''[a_1.@l_1, \ldots, a_n.@l_n, @l]]$. In this case, an XML tree T' is constructed from T as follows. For every $t \in \mathit{tuples}_D(T)$, define a tree tuple t' by using the following rule: $t'(q'.a'')$ is a fresh node identifier, $t'(q'.a''.@l) = t(q.@l)$, $t'(q'.a''.a_i)$ is a fresh node identifier ($i \in [1, n]$), $t'(q.a''.q_i.@l_i) = t(q_i.@l_i)$ and for every $q'' \in \mathit{paths}(D) - \{q.@l\}$, $t'(q'') = t(q'')$. Then T' is an XML tree whose tree tuples are $\{t' \mid t \in \mathit{tuples}_D(T)\}$. Furthermore, positions in t' are associated to positions in t as

follows. If $p' = (t'(q'.a''), @l)$, then $\pi_{T',T}(p') = \{(t(q), @l)\}$. If $p' = (t'(q'.a''.a_i), @l_i)$, then $(t(q_i), @l_i) \in \pi_{T',T}(p')$ (note that in this case $\pi_{T',T}(p)$ may contain more than one position). For any other position p' in t', $\pi_{T',T}(p') = \{p'\}$.

Similarly to the relational case, we can now show the following.

Theorem 7.5.9 *Let T be an XML tree that conforms to a DTD D and satisfies a set of FDs Σ, and let $T' \in inst(D', \Sigma')$ result from T by applying one step of the normalization algorithm. Let $p' \in Pos(T')$. Then*

$$\text{INF}_{T'}(p' \mid \Sigma') \quad \geq \quad \max_{p \in \pi_{T',T}(p')} \text{INF}_T(p \mid \Sigma).$$

PROOF: Let (D, Σ) be an XML specification and $T \in inst(D, \Sigma)$. Assume that (D, Σ) is not in XNF. Let (D', Σ') be an XML specification obtained by executing one step of the normalization algorithm. We have to prove that for every $p' \in Pos(T')$, $\text{INF}_{T'}(p' \mid \Sigma') \geq \max_{p\in\pi_{T',T}(p')} \text{INF}_T(p \mid \Sigma)$. This can be done in exactly the same way as the proof of Theorem 3.5.1. First, by using the same proof as for Lemma 3.5.2, we show that the same results holds for XML trees. Using this, we show the following:

1) Assume $D' = D[q.@l := q'.@m]$ and $q' \to q.@l$ is an anomalous FD over (D, Σ). Let a' be the last element of q' and $p' \in Pos(T')$. If p' is of the form $(x, @m)$, where $att(x, @m) = a'$, then $\text{INF}_{T'}(p' \mid \Sigma') = 1$ and, therefore, the theorem trivially holds. Otherwise, $\pi_{T',T}(p') = \{p'\}$ and it can be shown that $\text{INF}_{T'}(p' \mid \Sigma') \geq \text{INF}_T(p' \mid \Sigma)$ by using the same proof as that of Lemma 3.5.3.

2) Assume that $D' = D[q.@l := q'.a''[a_1.@l_1, \ldots, a_n.@l_n, @l]]$ and $\{q', q_1.@l_1, \ldots, q_n.@l_n\} \to q.@l$ is a (D, Σ)-minimal anomalous FD. Let $p' \in Pos(T')$. If p' is the position in T' of some value reachable from the root by following path $q'.a''.@l$ or $q'.a''.a_i.@l_i$, for some $i \in [1, n]$, then $\text{INF}_{T'}(p' \mid \Sigma') = 1$ since $\{q', q_1.@l_1, \ldots, q_n.@l_n\} \to q.@l$ is (D, Σ)-minimal. Thus, in this case the theorem trivially holds. Otherwise, $\pi_{T',T}(p') = \{p'\}$ and again it can be shown that $\text{INF}_{T'}(p' \mid \Sigma') \geq \text{INF}_T(p' \mid \Sigma)$ by using the same proof as for Lemma 3.5.3.

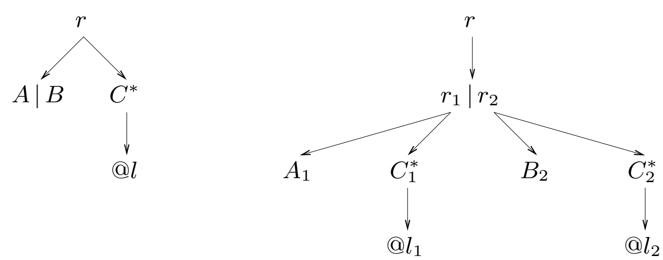
This completes the proof of the theorem. □

Just like in the relational case, one can define effective steps of the algorithm as those in which the above inequality is strict for at least one position, and show that (D, Σ) is in XNF if and only if no decomposition algorithm is effective in (D, Σ).

7.5.4 Eliminating additional assumptions

Finally, we have to show how to get rid of the additional assumption that for every anomalous FD $X \to p.@l$, every time that $p.@l$ is not null, every path in X is not null. We illustrate this by a simple example.

Assume that D is the DTD shown in Figure 7.5 (a). Every XML tree conforming to this DTD has as root an element of type r which has a child of type either A or B and an arbitrary number of elements of type C, each of them containing an attribute $@l$. Let Σ be the set of FDs $\{r.A \to r.C.@l\}$. Then, (D, Σ) is not in XNF since $(D, \Sigma) \not\vdash r.A \to r.C$.



(a) (b)

Figure 7.5: Splitting a DTD.

If we want to eliminate the anomalous FD $r.A \to r.C.@l$, we cannot directly apply the algorithm presented in Section 7.5.1, since this FD does not satisfy the basic assumption made in that section; it could be the case that $r.C.@l$ is not null and $r.A$ is null. To solve this problem we transform (D, Σ) into a new XML specification (D', Σ') that is essentially equivalent to (D, Σ) and satisfies the assumption made in Section 7.5.1. The new XML specification is constructed by splitting the disjunction. More precisely, DTD D' is defined as the DTD shown in Figure 7.5 (b). This DTD contains two copies of the DTD D, one of then containing element type A, denoted by A_1, and the other one containing element type B, denoted by B_2. The set of functional dependencies Σ' is constructed by including the FD $r.A \to r.C.@l$ in both DTDs, that is, $\Sigma' = \{r.A_1 \to r.C_1.@l_1,\ r.A_2 \to r.C_2.@l_2\}$.

In the new specification (D', Σ'), the user chooses between having either A or B by choosing between either r_1 or r_2. We note that the new FD $r.A_2 \to r.C_2.@l_2$ is trivial and, therefore, to normalize the new specification we only have to take into account FD $r.A_1 \to r.C_1.@l_1$. This functional dependency satisfies the assumption made in Section 7.5.1, so we can use the decomposition algorithm presented in that section.

It is straightforward to generalize the methodology presented in the previous example for any DTD. In particular, if we have an arbitrary regular expression s in a DTD $D = (E, A, P, R, r)$ and we have to split it into one regular expression containing an element type $\tau \in E$ and another one not containing this symbol, we consider regular expressions $s \cap (E^*\tau E^*)$ and $s - (E^*\tau E^*)$.

7.6 A Third Normal Form for XML

In Section 2.1, we show that BCNF has the advantage of eliminating redundant information and has the disadvantage of requiring certain functional dependencies to be maintained only as inter-relational constraints. Thus, sometimes in practice (especially in those cases where enforcing integrity of the database is crucial) one aims at 3NF since the decompositions based on 3NF are dependency preserving.

In Section 7.2.1, we have shown a direct mapping of relational databases into XML where there exists a one-to-one correspondence between functional dependencies in these two models. For XML specifications generated by this mapping, the XNF decomposition algorithm works as the BCNF decomposition algorithm and, thus, the former algorithm cannot be dependency preserving. Even though we have not proved that XNF is not dependency preserving, we strongly believe that this is the case. Hence, the situation for XML is similar to the one for BCNF; XNF eliminates redundant information but it is not dependency preserving in general.

A natural question at this point is how does a third normal form for XML look like. Although in this dissertation we have not studied 3NF for XML, we have given all the necessary components to define such a normal form: an XML functional dependency language and a syntactic condition to extend BCNF to XML. In fact, in [Kol05] Kolahi uses these two elements to propose a third normal form for XML (X3NF). We give here a brief description of X3NF.

To extend 3NF to XML, Kolahi extends to the notion of prime attribute (see Section 2.1.3) to the case of paths. More precisely, a path $p.@l$ is a *prime path* if there exists a nontrivial FD $S \to q \in (D, \Sigma)^+$ such that q is an element path, $p.@l \in S$ and $S - \{p.@l\} \to q \notin (D, \Sigma)^+$. Then an XML specification (D, Σ) is in X3NF if and only if for every nontrivial FD $S \to p.@l \in (D, \Sigma)^+$, we have that $S \to p \in (D, \Sigma)^+$ or $p.@l$ is a prime path [Kol05]. It remains to be proved that the decompositions based on X3NF definition are dependency preserving.

7.7 Related Work

Embley and Mok [EM01a] introduced an XML normal form defined in terms of functional dependencies, multi-valued dependencies and inclusion constraints. Although that normal form was also called XNF, the approach of [EM01a] was very different from ours. The normal form of [EM01a] was defined in terms of two conditions: XML specifications must not contain redundant information with respect to a set of constraints, and the number of schema trees (see Section 7.2.2) must be minimal. The normalization process is similar to the ER approach in relational databases. A conceptual-model hypergraph is constructed to model the real world and an algorithm produces an XML specification in XNF. It is proved in Section 7.4 that an XML specification given by a DTD D and a set Σ of XML functional dependencies is in XNF if and only if no XML tree conforming to D and satisfying Σ contains redundant information. Thus, for the class of functional dependencies defined in this chapter, the XML normal form introduced in [EM01a] is more restrictive than our XML normal form.

Normal forms for extended context-free grammars, similar to the Greibach normal form for CFGs, were considered in [AGW01]. These, however, do not necessarily guarantee good XML design.

Lee et al. [LLL02] introduced a functional dependency language for XML (see Section 6.5 for a precise description). The normalization problem is not considered in this paper.

Chapter 8

Conclusions

The goal of this dissertation was to find principles for good XML data design, and algorithms to produce such designs. Seeking for such principles, we realized that while in the relational world the criteria for being well designed are usually very intuitive and clear to state, they become more obscure when one moves to more complex data models such as XML. Thus, our first task was to find criteria for good data design based on the intrinsic properties of a data model rather than tools built on top of it, such as query and update languages. We were motivated by the justification of normal forms for XML, where usual criteria based on update anomalies or existence of lossless decompositions are not applicable until we have standard and universally accepted query and update languages. We proposed to use techniques from information theory, and we developed a measure of information content of elements in a database with respect to a set of data dependencies.

This information-theoretic measure is the main contribution of this dissertation. As in the case of relational databases, principles for good XML data design are expressed as normal forms that well-designed databases are expected to satisfy. As such, normal forms play a central role in the design of XML databases. The information-theoretic measure proposed in this dissertation is a general and robust tool that can used in different ways to study normal forms in data models such as the relational model and XML. The following are some applications of this tool.

- First, it can be employed to justify normal forms. In fact, in this dissertation we showed that it characterizes well-known relational normal forms such as BCNF and 4NF as precisely those corresponding to good designs; furthermore, it justifies others, more complicated ones, involving join dependencies. We then showed that for the case of constraints given by XML functional dependencies, it equates the XML normal form XNF –proposed in this dissertation– with good designs.

- Second, the information-theoretic measure can be used to justify normalization algorithms. Indeed, in this dissertation we looked at information-theoretic justifications for normalization algorithms for relational and XML databases.

- Third, our measure can be employed to aid in the process of finding a normal form for a class of data dependencies. Recall that, as in the case of relational databases, the design of XML databases is guided by the semantic information encoded in data dependencies. As such, normal forms are usually defined as syntactic conditions on classes of data dependencies. In general, finding these syntactic conditions is a nontrivial problem, especially in XML. Our information-theoretic approach offers a simple solution to this problem. Regardless of how complicated is an XML data dependency language, the information-theoretic approach can be immediately used to provide a normal form for this class of dependencies. Thus, even if no syntactic normal form is known for a class of data dependencies, we can still check whether an XML database containing this type of dependencies is well-designed.

Certainly, the information-theoretic approach proposed in this dissertation does not solve all the problems related to the design of XML databases. In particular, once a normal form has been justified by this approach, there are two additional problems that need to be solved to make it practical: testing whether a database is in this normal form and transforming a database into an equivalent one in this normal form.

In this dissertation, we propose a language for XML functional dependencies and a normal form, XNF, for XML databases containing this type of dependencies. Since our goal was to find algorithms to produce good designs, apart from providing an information-theoretic justification for XNF, we also investigate the complexity of testing XNF and transforming an XML database into one in XNF. More specifically, we identify some natural cases whether this problem can be solved efficiently and we present an algorithm for converting any XML schema into an equivalent one in XNF. Thus, the second main contribution of this dissertation is to have shown that XNF can be used in practice, since (1) the functional dependency language used in XNF is simple and expressive, (2) in most practical cases we can test XNF efficiently, and (3) there exists an algorithm for transforming any XML schema into an equivalent one in XNF.

Chapter 9

Future Work

The following is a list of problems for future research.

- In this dissertation, we propose an information-theoretic measure that takes into account both instance and schema constraints, unlike the measures studied before [Lee87, CP87, DR00, LL03].

  Our information-theoretic approach can be used to measure the information content of a position of an instance. It would be interesting to take a step forward, and define an information-theoretic measure that can be used to reason about database schemas. In particular, it would be interesting to connect this new approach with that of Hull [Hul86], where information capacities of two relational schemas can be compared based on the existence of queries in some standard language that translates between them. For two classes of well-designed schemas (those with no constraints, and with keys only), being information-capacity equivalent means being identical [AIR99, Hul86] (up to renaming and re-ordering of attributes and relations), and we would like to see if this connection extends beyond the classes of schemas studied by Hull [Hul86] and Albert et al. [AIR99].

- It is an interesting problem for future research to extend the set-theoretic measure presented in Section 2.1.4 to reason about normalization algorithms and normal forms that allow redundant information.

- It would be interesting to characterize 3NF by using our information-theoretic measure. So far, a little is known about 3NF. For example, as in the case of BCNF, it is possible to prove that the synthesis approach for generating 3NF databases does not decrease the amount of information in each position. Furthermore, given that 3NF does not necessarily eliminate all redundancies, one can find 3NF databases where the amount of information in some positions is not maximal.

- It remains as an open problem what is the exact complexity of the functional dependency implication problem. In this dissertation, we prove that this problem is NP-hard, and can be solved in co-NEXPTIME, and we also identify some classes of DTDs for which this problem can be solved efficiently. It would be interesting to close the complexity gap and to identify other natural classes of DTDs for which this problem is tractable.

- As prevalent as BCNF is, it does not solve all the problems of relational schema design, and one cannot expect XNF to address all shortcomings of DTD design. It would be interesting to extend XNF to more powerful normal forms, in particular by taking into account more expressive functional dependency languages and multivalued dependencies, which are naturally induced by the tree structure of XML documents.

- The XNF decomposition algorithm introduced in Section 7.5.1 can be improved in various ways. In particular, it would be interesting to work on making it more efficient.

- Finally, it would be interesting to study the complexity of checking consistency for more complex XML constraints, e.g., those defined in terms of XPath [CD], and more complex schema specifications such as the type system of XQuery [BCF+]. In particular, it would be interesting to considered the case of extended DTDs [PV00], which precisely characterize unranked tree automata [Tha67, BKMW01]. Our lower bounds apply to those settings, but it is open whether upper bounds remain intact.

Bibliography

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 2000.

[ABU79] Alfred Aho, Catriel Beeri, and Jeffrey D. Ullman. The Theory of Joins in Relational Databases. *ACM Transactions on Database Systems*, 4(3):297–314, 1979.

[AFL02a] Marcelo Arenas, Wenfei Fan, and Leonid Libkin. On Verifying Consistency of XML Specifications. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 259–270, 2002.

[AFL02b] Marcelo Arenas, Wenfei Fan, and Leonid Libkin. What's Hard about XML Schema Constraints? In *Proceedings of the 13th International Conference on Database and Expert Systems Applications*, pages 269–278, 2002.

[AGW01] Jurgen Albert, Dora Giammarresi, and Derick Wood. Normal Form Algorithms for Extended Context-free Grammars. *Theoretical Computer Science*, 267(1-2):35 – 47, 2001.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AIR99] Joseph Albert, Yannis Ioannidis, and Raghu Ramakrishnan. Equivalence of Keyed Relational Schemas by Conjunctive Queries. *Journal of Computer and System Sciences*, 58(3):512–534, 1999.

[AL02] Marcelo Arenas and Leonid Libkin. A Normal Form for XML Documents. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 85–96, 2002.

[AL03] Marcelo Arenas and Leonid Libkin. An Information-Theoretic Approach to Normal Forms for Relational and XML Data. In *Proceedings of the Twenty-second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 15 – 26, 2003.

[AL04] Marcelo Arenas and Leonid Libkin. A Normal Form for XML Documents. *ACM Transactions on Database Systems*, 29(1):195–232, 2004.

[AL05] Marcelo Arenas and Leonid Libkin. An Information-Theoretic Approach to Normal Forms for Relational and XML Data. *Journal of the ACM*, 52(2):246–283, 2005.

[AM84] Paolo Atzeni and Nicola Morfuni. Functional Dependencies in Relations with Null Values. *Information Porcessing Letters*, 18(4):233–238, 1984.

[Arm74] W. W. Armstrong. Dependency Structures of Data Base Relationships. In *Proceedings of the IFIP Congress 74*, pages 580–583, 1974.

[ASV01] Serge Abiteboul, Luc Segoufin, and Victor Vianu. Representing and Querying XML with Incomplete Information. In *Proceedings of the Twentieth ACM Symposium on Principles of Database Systems*, pages 150 – 161, 2001.

[AV99] Serge Abiteboul and Victor Vianu. Regular Path Queries with Constraints. *Journal of Computer and System Sciences*, 58(3):428–452, 1999.

[BB79] Catriel Beeri and Philip Bernstein. Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Transactions on Database Systems*, 4(1):30–59, 1979.

[BBG78] Catriel Beeri, Philip Bernstein, and Nathan Goodman. A Sophisticate's Introduction to Database Normalization Theory. In *Fourth International Conference on Very Large Data Bases*, pages 113–124, 1978.

[BCF$^+$] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. W3C Working Draft, April 2005. `http://www.w3.org/TR/xquery`.

[BCF$^+$03] Michael Benedikt, Chee Yong Chan, Wenfei Fan, Juliana Freire, and Rajeev Rastogi. Capturing both Types and Constraints in Data Integration. In

Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, pages 277–288, 2003.

[BDB79] Joachim Biskup, Umeshwar Dayal, and Philip Bernstein. Synthesizing Independent Database Schemas. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, pages 143–151, 1979.

[BDF+01a] Peter Buneman, Susan Davidson, Wenfei Fan, Carmem Hara, and Wang Chiew Tan. Keys for XML. In *Proceedings of the Tenth International World Wide Web Conference*, pages 201–210, 2001.

[BDF+01b] Peter Buneman, Susan Davidson, Wenfei Fan, Carmem Hara, and Wang Chiew Tan. Reasoning about Keys for XML. In *Proceedings of the Eighth International Workshop on Database Programming Languages*, 2001.

[BDF+02] Peter Buneman, Susan B. Davidson, Wenfei Fan, Carmem S. Hara, and Wang Chiew Tan. Keys for XML. *Computer Networks*, 39(5):473–487, 2002.

[BDF+03] Peter Buneman, Susan B. Davidson, Wenfei Fan, Carmem S. Hara, and Wang Chiew Tan. Reasoning about Keys for XML. *Information Systems*, 28(8):1037–1063, 2003.

[Bee80] Catriel Beeri. On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases. *ACM Transactions on Database Systems*, 5(3):241–259, 1980.

[Ber76] Philip Bernstein. Synthesizing Third Normal Form Relations from Functional Dependencies. *ACM Transactions on Database Systems*, 1(4):277–298, 1976.

[Ber79] Philip Bernstein. Errata: Computational Problems Related to the Design of Normal Form Relational Schemas. *ACM Transactions on Database Systems*, 4(3):396, 1979.

[BFH77] Catriel Beeri, Ronald Fagin, and John Howard. A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations. In *Proceedings of the 1977 ACM SIGMOD International Conference on Management of Data*, pages 47–61. ACM, 1977.

[BFMY83] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the Desirability of Acyclic Database Schemes. *Journal of the ACM*, 30(3):479–513, 1983.

[BFSW01] Peter Buneman, Wenfei Fan, Jérôme Siméon, and Scott Weinstein. Constraints for Semi-structured Data and XML. *SIGMOD Record*, 30(1):47–45, 2001.

[BFW98] Peter Buneman, Wenfei Fan, and Scott Weinstein. Path Constraints in Semistructured and Structured Databases. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 129–138, 1998.

[BG80] Philip Bernstein and Nathan Goodman. What does Boyce-Codd Normal Form Do? In *Sixth International Conference on Very Large Data Bases*, pages 245–259, 1980.

[BGL+99] Chaitanya K. Baru, Amarnath Gupta, Bertram Ludäscher, Richard Marciano, Yannis Papakonstantinou, Pavel Velikhov, and Vincent Chu. XML-Based Information Mediation with MIX. In *Proceedings of th 1999 ACM SIGMOD International Conference on Management of Data*, pages 597–599, 1999.

[BJO91] Peter Buneman, Achim Jung, and Atsushi Ohori. Using Powerdomains to Generalize Relational Databases. *Theoretical Computer Science*, 91(1):23–55, 1991.

[BKMW01] Anne Bruüggemann-Klein, Makoto Murata, and Derick Wood. Regular Tree and Regular Hedge Languages over Unranked Alphabets. HKUST Theoretical Computer Science Center Research Report; HKUST-TCSC-2001-05, 2001.

[BM99] Catriel Beeri and Tova Milo. Schemas for Integration and Translation of Structured and Semi-structured Data. In *Proceedings of the 7th International Conference on Database Theory*, pages 296–313, 1999.

[BNdB04] Geert Jan Bex, Frank Neven, and Jan Van den Bussche. DTDs versus XML Schema: A Practical Study. In *Proceedings of the Seventh International Workshop on the Web and Databases*, pages 79–84, 2004.

[Buf93] H. W. Buff. Remarks on Two New Theorems of Date and Fagin. *SIGMOD Record*, 22(1):55–56, 1993.

[CD] James Clark and Steve DeRose. XML Path Language (XPath) Version 1.0. W3C Recommendation, November 1999. `http://www.w3.org/TR/xpath`.

[CFI+00] Michael J. Carey, Daniela Florescu, Zachary G. Ives, Ying Lu, Jayavel Shanmugasundaram, Eugene J. Shekita, and Subbu N. Subramanian. XPERANTO: Publishing Object-Relational Data as XML. In *Proceedings of the Third International Workshop on the Web and Databases*, pages 105–110, 2000.

[CFP84] Marco Casanova, Ronald Fagin, and Christos Papadimitriou. Inclusion Dependencies and Their Interaction with Functional Dependencies. *Journal of Computer and System Sciences*, 28(1):29–59, 1984.

[CGL99] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Representing and Reasoning on XML Documents: A Description Logic Approach. *Journal of Logic and Computation*, 9(3):295–318, 1999.

[Cho02] Byron Choi. What are real DTDs like? In *WebDB*, pages 43–48, 2002.

[CKV90] Stavros S. Cosmadakis, Paris C. Kanellakis, and Moshe Y. Vardi. Polynomial-Time Implication Problems for Unary Inclusion Dependencies. *Journal of the ACM*, 37(1):15–46, 1990.

[Cla] James Clark. XSL transformations (XSLT) version 1.0. w3c recommendation, november 1999. `http://www.w3.org/TR/xslt`.

[Cod70] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Cod72] E. F. Codd. Further Normalization of the Data Base Relational Model. In *Data base systems*, pages 33–64. Englewood Cliffs, N.J. Prentice-Hall, 1972.

[Cod74] E. F. Codd. Recent Investigations in Relational Data Base Systems. In *IFIP Congress*, pages 1017–1021, 1974.

[CP87] Roger Cavallo and Michael Pittarelli. The Theory of Probabilistic Databases. In *Proceedings of 13th International Conference on Very Large Data Bases*, pages 71–81, 1987.

[CT91] Thomas Cover and Joy Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

[CV85] Ashok Chandra and Moshe Vardi. The Implication Problem for Functional and Inclusion Dependencies is Undecidable. *SIAM Journal on Computing*, 14(3):671–677, 1985.

[DF92] C. J. Date and Ronald Fagin. Simple Conditions for Guaranteeing Higher Normal Forms in Relational Databases. *ACM Transactions on Database Systems*, 17(3):465–476, 1992.

[DF93] C. J. Date and Ronald Fagin. Response to "Remarks on Two New Theorems of Date and Fagin". *SIGMOD Record*, 22(1):57–58, 1993.

[DG84] William Dowling and Jean Gallier. Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae. *Journal of Logic Programming*, 1(3):267–284, 1984.

[DGV99] Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. Improved Automata Generation for Linear Temporal Logic. In *Proceedings of the 11th International Conference on Computer Aided Verification*, pages 249–260, 1999.

[DR00] Mehmet Dalkilic and Edward Robertson. Information Dependencies. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 245–253, 2000.

[ebX] ebXML. Business Process Specification Schema v1.01. `http://www.ebxml.org/specs/`.

[EM01a] David Embley and Wai Yin Mok. Developing XML Documents with Guaranteed "Good" Properties. In *Proceedings of the Twentieth International Conference on Conceptual Modeling*, pages 426–441, 2001.

[EM01b] Anat Eyal and Tova Milo. Integrating and Customizing Heterogeneous E-commerce Applications. *The VLDB Journal*, 10(1):16–38, 2001.

[EN99] Ramez Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 3rd edition, 1999.

[Fag77] Ronald Fagin. Multivalued Dependencies and a New Normal Form for Relational Databases. *ACM Transactions on Database Systems*, 2(3):262–278, 1977.

[Fag79] Ronald Fagin. Normal Forms and Relational Database Operators. In *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, pages 153–160. ACM, 1979.

[Fag81] Ronald Fagin. A Normal Form for Relational Databases That Is Based on Domians and Keys. *ACM Transactions on Database Systems*, 6(3):387–415, 1981.

[FK99] Daniela Florescu and Donald Kossmann. Storing and Querying XML Data using an RDMBS. *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[FL01] Wenfei Fan and Leonid Libkin. On XML Integrity Constraints in the Presence of DTDs. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 114 – 125, 2001.

[FL02] Wenfei Fan and Leonid Libkin. On XML integrity constraints in the presence of DTDs. *Journal of the ACM*, 49(3):368–406, 2002.

[FMU82] Ronald Fagin, Alberto Mendelzon, and Jeffrey Ullman. A Simplified Universal Relation Assumption and Its Properties. *ACM Transactions on Database Systems*, 7(3):343–360, 1982.

[FS00] Wenfei Fan and Jérôme Siméon. Integrity Constraints for XML. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 23–34, 2000.

[FS03] Wenfei Fan and Jérôme Siméon. Integrity constraints for XML. *Journal of Computer and System Sciences*, 66(1):254–291, 2003.

[FSTG85] Patrick Fischer, Lawrence Saxton, Stan Thomas, and Dirk Van Gucht. Interactions between Dependencies and Nested Relational Structures. *Journal of Computer and System Sciences*, 31(3):343–354, 1985.

[FT83] Patrick Fischer and Don-Min Tsou. Whether a Set of Multivalued Dependencies Implies a Join Dependency is NP-Hard. *SIAM Journal on Computing*, 12(2):259–266, 1983.

[FV86] Ronald Fagin and Moshe Vardi. The Theory of Data Dependencies: a Survey. *Mathematics of Information Processing, Proceedings of Symposia in Applied Mathematics, American Mathematical Society*, 34:19–72, 1986.

[Gal82] Zvi Galil. An Almost Linear-Time Algorithm for Computing a Dependency Basis in a Relational Database. *Journal of the ACM*, 29(1):96–102, 1982.

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[GMUW01] Hector Garcia-Molina, Jeffrey Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall, 1st edition, 2001.

[GMWK02] Robert Givan, David A. McAllester, Carl Witty, and Dexter Kozen. Tarskian Set Constraints. *Information and Computation*, 174(2):105–131, 2002.

[GN02] Evgueni Goldberg and Yakov Novikov. BerkMin: A Fast and Robust Sat-Solver. In *Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exposition*, pages 142–149, 2002.

[Gol91] Charles F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1991.

[GR83] Gösta Grahne and Kari-Jouko Räihä. Database Decomposition into Fourth Normal Form. In *9th International Conference on Very Large Data Bases*, pages 186–196, 1983.

[Gra91] Gosta Grahne. *The Problem of Incomplete Information in Relational Databases*. Springer, 1991.

[Gun92] Carl Gunter. *Semantics of Programming Languages: Structures and Techniques*. MIT Press, 1992.

[HD99] Carmem Hara and Susan Davidson. Reasoning about Nested Functional Dependencies. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 91–100, 1999.

[HJ99] Justin Higgins and Rick Jelliffe. QAML Version 2.4. `http://xml.ascc.net/resource/qaml-xml.dtd`, 1999.

[Hol03] Gerard J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, 2003.

[HU79] John Hopcroft and Jeffrey Ullman. *Introduction to Automata Theory*. Addison-Wesley, 1979.

[Hul86] Richard Hull. Relative Information Capacity of Simple Relational Database Schemata. *SIAM Journal on Computing*, 15(3):856–886, 1986.

[Hun00] David Hunter. *Beginning XML*. Wrox Press, 1st edition, 2000.

[IJ84] Tomasz Imielinski and Witold Lipski Jr. Incomplete Information in Relational Databases. *Journal of the ACM*, 31(4):761–791, 1984.

[JF82] Jiann Jou and Patrick Fischer. The Complexity of Recognizing 3NF Relation Schemes. *Information Processing Letters*, 14(4):187–190, 1982.

[Kan90] Paris Kanellakis. Elements of Relational Database Theory. In *Handbook of Theoretical Computer Science*, Volume B, pages 1075-1144, MIT Press, 1990.

[KM00] Carl-Christian Kanne and Guido Moerkotte. Efficient Storage of XML Data. In *Proceedings of the 16th International Conference on Data Engineering*, page 198, 2000.

[Kol05] Solmaz Kolahi. Dependency-Preserving Normalization of Relational and XML Data. In *Proceedings of the 10th International Symposium on Database Programming Languages*, 2005.

[LC00] Dongwon Lee and Wesley W. Chu. Constraints-Preserving Transformation from XML Document Type Definition to Relational Schema. In *Proceedings of the 19th International Conference on Conceptual Modeling*, pages 323–338, 2000.

[Lee87] Tony Lee. An Information-Theoretic Analysis of Relational Databases - Part I: Data Dependencies and Information Metric. *IEEE Transactions on Software Engineering*, 13(10):1049–1061, 1987.

[Ley] Michael Ley. DBLP. `http://www.informatik.uni-trier.de/~ley/db/index.html`.

[LJ82] Carol Helfgott LeDoux and Douglas Stott Parker Jr. Reflections on Boyce-Codd Normal Form. In *Eigth International Conference on Very Large Data Bases*, pages 131–141, 1982.

[LJM+] Andrew Layman, Edward Jung, Eve Maler, Henry S. Thompson, Jean Paoli, John Tigue, Norbert H. Mikula, and Steve De Rose. XML-Data. W3C Note, January 1998. `http://www.w3.org/TR/1998/NOTE-XML-data-0105/`.

[LL98] Mark Levene and George Loizou. Axiomatisation of Functional Dependencies in Incomplete Relations. *Theoretical Computer Science*, 206(1-2):283–300, 1998.

[LL03] Mark Levene and George Loizou. Why is the Snowflake Schema a Good Data Warehouse Design? *Information Systems*, 28(3):225–240, 2003.

[LLD04] Tok Wang Ling, Mong-Li Lee, and Gillian Dobbie. *Semistructured Database Design*. Springer, 2004.

[LLL02] Mong-Li Lee, Tok Wang Ling, and Wai Lup Low. Designing Functional Dependencies for XML. In *Proceedings of the Eighth International Conference on Extending Database Technology*, pages 124–141, 2002.

[Mak77] Akifumi Makinouchi. A Consideration on Normal Form of Not-Necessarily-Normalized Relation in the Relational Data Model. In *Proceedings of the Third International Conference on Very Large Data Bases*, pages 447–453, 1977.

[Mat93] Yuri Matiyasevich. *Hilbert's 10th Problem*. MIT Press, 1993.

[Men79] Alberto Mendelzon. On Axiomatizing Multivalued Dependencies in Relational Databases. *Journal of the ACM*, 26(1):37–44, 1979.

[Mit83] John Mitchell. The Implication Problem for Functional and Inclusion Dependencies. *Information and Control*, 56(3):154–173, 1983.

[MMS79] David Maier, Alberto Mendelzon, and Yehoshua Sagiv. Testing Implications of Data Dependencies. *ACM Transactions on Database Systems*, 4(4):455–469, 1979.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference*, pages 530–535, 2001.

[MNE96] Wai Yin Mok, Yiu-Kai Ng, and David Embley. A Normal Form for Precisely Characterizing Redundancy in Nested Relations. *ACM Transactions on Database Systems*, 21(1):77–106, 1996.

[Mok02] Wai Yin Mok. A Comparative Study of Various Nested Normal Forms. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):369–385, 2002.

[MR89] Heikki Mannila and Kari-Jouko Räihä. Practical Algorithms for Finding Prime Attributes and Testing Normal Forms. In *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 128–133, 1989.

[MS93] Jim Melton and Alan R. Simon. *Understanding the New SQL: A Complete Guide*. Morgan Kaufmann, 1993.

[MSY81] David Maier, Yehoshua Sagiv, and Mihalis Yannakakis. On the Complexity of Testing Implications of Functional and Join Dependencies. *Journal of the ACM*, 28(4):680–695, 1981.

[Nev99] Frank Neven. Extensions of Attribute Grammars for Structured Document Queries. In *Proceedings of the 7th International Workshop on Database Programming Languages*, pages 99–116, 1999.

[Nev02] Frank Neven. Automata Theory for XML Researchers. *SIGMOD Record*, 31(3):39–46, 2002.

[ÖY87] Meral Özsoyoglu and Li-Yan Yuan. A New Normal Form for Nested Relations. *ACM Transactions on Database Systems*, 12(1):111–136, 1987.

[ÖY89] Meral Özsoyoglu and Li-Yan Yuan. On the Normalization in Nested Relational Databases. In *Nested Relations and Complex Objects*, pages 243–271. Springer, 1989.

[Pap81] Christos Papadimitriou. On the Complexity of Integer Programming. *Journal of the ACM*, 28(4):765–768, 1981.

[Pap94] Christos Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[PBGG89] Jan Paredaens, Paul De Bra, Marc Gyssens, and Dirk Van Gucht. *The Structure of the Relational Database Model*, chapter 5, pages 132–155. Springer-Verlag, 1989.

[PDST00] Lucian Popa, Alin Deutsch, Arnaud Sahuguet, and Val Tannen. A chase too far? In *SIGMOD Conference*, pages 273–284, 2000.

[Pet89] Sergey Petrov. Finite Axiomatisation of Languages for Representation of System Properties: Axiomatization of Dependencies. *Information Sciences*, 47(3):339–372, 1989.

[PV00] Yannis Papakonstantinou and Victor Vianu. DTD Inference for Views of XML Data. In *Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 35–46, 2000.

[RKS88] Mark Roth, Henry Korth, and Abraham Silberschatz. Extended Algebra and Calculus for Nested Relational Databases. *ACM Transactions on Database Systems*, 13(4):389–417, 1988.

[Sci81] Edward Sciore. Real-World MVD's. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, pages 121–132, 1981.

[SDPF81] Yehoshua Sagiv, Claude Delobel, D. Scott Parker, and Ronald Fagin. An Equivalence Between Relational Database Dependencies and a Fragment of Propositional Logic. *Jornal of the ACM*, 28(3):435–453, 1981.

[Sha48] Claude Shannon. A Mathematical Theory of Communication. *Bell System Technical Journal*, 27:379–423 (Part I), 623–656 (Part II), 1948.

[SMD03] Arijit Sengupta, Sriram Mohan, and Rahul Doshi. XER - Extensible Entity Relationship Modeling. In *Proceedings of the XML 2003 Conference*, 2003.

[SS82] Mario Schkolnick and Paul G. Sorenson. The Effects of Denormalisation on Database Performance. *Australian Computer Journal*, 14(1):12–18, 1982.

[SSB+00] Jayavel Shanmugasundaram, Eugene J. Shekita, Rimon Barr, Michael J. Carey, Bruce G. Lindsay, Hamid Pirahesh, and Berthold Reinwald. Efficiently Publishing Relational Data as XML Documents. In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 65–76, 2000.

[STZ+99] Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David DeWitt, and Jeffrey Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of 25th International Conference on Very Large Data Bases*, pages 302–314, 1999.

[Suc01] Dan Suciu. On Database Theory and XML. *SIGMOD Record*, 30(3):39–45, 2001.

[TBMM] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema. W3C Recommendation, October 2004. `http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/`.

[TF82] Don-Min Tsou and Patrick Fischer. Decomposition of a Relation Schema into Boyce-Codd Normal Form. *SIGACT News*, 14(3):23–29, 1982.

[Tha67] James W. Thatcher. Characterizing Derivation Trees of Context-Free Grammars through a Generalization of Finite Automata Theory. *Journal of Computer and System Sciences*, 1(4):317–322, 1967.

[TIHW01] Igor Tatarinov, Zachary Ives, Alon Halevy, and Daniel Weld. Updating XML. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, pages 413–424, 2001.

[Ull88] Jeffrey Ullman. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.

[Via01] Victor Vianu. A Web Odyssey: from Codd to XML. In *Proceedings of the Twenteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 1–15, 2001.

[Vin97] Millist Vincent. A Corrected 5NF Definition for Relational Database Design. *Theoretical Computer Science*, 185(2):379–391, 1997.

[Vin99] Millist Vincent. Semantic Foundations of 4NF in Relational Database Design. *Acta Informatica*, 36(3):173–213, 1999.

[VW94] Moshe Vardi and Pierre Wolper. Reasoning about Infinite Computations. *Information and Computation*, 115(1):1–37, 1994.

[Wed92] Grant Weddell. Reasoning about Functional Dependencies Generalized for Semantic Data Models. *ACM Transactions on Database Systems*, 17(1):32–64, 1992.

[Wid99] Jennifer Widom. Data Management for XML: Research Directions. *IEEE Data Engineering Bulletin*, 22(3):44–52, 1999.

[WLLD01] Xiaoying Wu, Tok Wang Ling, Mong-Li Lee, and Gillian Dobbie. Designing Semistructured Databases Using ORA-SS Model. In *Proceedings of the 2nd International Conference on Web Information Systems Engineering*, pages 171–180, 2001.

[YÖ86] Li-Yan Yuan and Meral Özsoyoglu. Unifying Functional and Multivalued Dependencies for Relational Database Design. In *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 183–190, 1986.

[YP04] Cong Yu and Lucian Popa. Constraint-based xml query rewriting for data integration. In *SIGMOD Conference*, pages 371–382, 2004.

[Zan82] Carlo Zaniolo. A New Normal Form for the Design of Relational Database Schemata. *ACM Transactions on Database Systems*, 7(3):489–499, 1982.

[ZM02] Lintao Zhang and Sharad Malik. The Quest for Efficient Boolean Satisfiability Solvers. In *Proceedings of the 14th International Conference on Computer Aided Verification*, pages 17–36, 2002.

Appendix A

Proofs from Chapter 3

A.1 Proof of Lemma 3.4.4

We start with the following simple but useful observation. The proof follows immediately from genericity.

Claim A.1.1 *Let Σ be a set of generic integrity constraints over a relational schema S, $I \in inst_k(S, \Sigma)$ and $p \in Pos(I)$. Assume that $a, b \in [1, k] - adom(I)$. Then for every $\bar{a} \in \Omega(I, p)$, $|SAT^k_\Sigma(I_{(a,\bar{a})})| = |SAT^k_\Sigma(I_{(b,\bar{a})})|$.*

Next, we need the following.

Claim A.1.2 *Let Σ be a set of integrity constraints over a relational schema S, $I \in inst(S, \Sigma)$, $p \in Pos(I)$ and $\bar{a} \in \Omega(I, p)$. Then for every $a \in \mathbb{N}^+$, there exists $k_0 \in \mathbb{N}^+$ and a polynomial $q_a(k)$ such that $|SAT^k_\Sigma(I_{(a,\bar{a})})| = q_a(k)$, for every $k > k_0$.*

PROOF: Let the variables of $\bar{a}$ be $v_1, \ldots, v_l$. Fix $a > 0$, and let m be the maximum value in $adom(I) \cup \{a\}$. Define k_0 to be $m + l + 1$. By genericity, $|SAT^{k_0}_\Sigma(I_{(a,\bar{a})})| = 0$ implies $|SAT^k_\Sigma(I_{(a,\bar{a})})| = 0$ for all $k > k_0$, so we assume there is at least one substitution in $SAT^{k_0}_\Sigma(I_{(a,\bar{a})})$.

We consider the set of all triples $\mathcal{P} = (X, \sigma_X, \Pi)$ where

- $X \subseteq \{1, \ldots, l\}$,
- $\sigma_X : \{v_i \mid i \in X\} \to [1, m]$, and
- Π is a partition on $\{1, \ldots, l\} - X$.

Given $\sigma \in SAT^k_\Sigma(I_{(a,\bar{a})})$, we write $\sigma \sim \mathcal{P}$ if for every $i \in X$, $\sigma(v_i) = \sigma_X(v_i)$, for every $i \notin X$, $\sigma(v_i) \notin [1,m]$, and for every $i, j \notin X$, $\sigma(v_i) = \sigma(v_j)$ iff i and j are in the same block of Π. Observe that for every $\sigma \in SAT^k_\Sigma(I_{(a,\bar{a})})$, there exists exactly one triple $\mathcal{P}$ such that $\sigma \sim \mathcal{P}$.

Let $\sigma, \sigma' \sim \mathcal{P}$ be two substitutions. From the genericity of Σ we immediately see that $\sigma(I_{(a,\bar{a})}) \models \Sigma$ iff $\sigma'(I_{(a,\bar{a})}) \models \Sigma$. Furthermore, if σ collapses two rows in $I_{(a,\bar{a})}$, then so does σ' (since $\sigma(v_i) = \sigma(v_j)$ iff $\sigma'(v_i) = \sigma'(v_j)$). We conclude that $\sigma \in SAT^k_\Sigma(I_{(a,\bar{a})})$ iff $\sigma' \in SAT^k_\Sigma(I_{(a,\bar{a})})$.

The number of triples $\mathcal{P}$ depends on I, a and $\bar{a}$ but not on k. For each $\mathcal{P}$, either all σ with $\sigma \sim \mathcal{P}$ belong to $SAT^k_\Sigma(I_{(a,\bar{a})})$, or none belongs to $SAT^k_\Sigma(I_{(a,\bar{a})})$. Thus, it will suffice to show that for every $\mathcal{P}$, there exists a polynomial $q^{\mathcal{P}}_a(k)$ such that $|\{\sigma \in SAT^k_\Sigma(I_{(a,\bar{a})}) \mid \sigma \sim \mathcal{P}\}| = q^{\mathcal{P}}_a(k)$.

The case when no σ with $\sigma \sim \mathcal{P}$ belongs to $SAT^k_\Sigma(I_{(a,\bar{a})})$ is trivial: $q^{\mathcal{P}}_a(k) = 0$ for all k. Otherwise, let $\mathcal{P} = (X, \sigma_X, \Pi)$, and let $m_{\mathcal{P}}$ be the number of partition blocks of Π. The number of $\sigma \sim \mathcal{P}$ is then the number of ways to chose $m_{\mathcal{P}}$ distinct ordered elements in $[m+1, k]$, that is

$$q^{\mathcal{P}}_a(k) \quad = \quad \prod_{i=0}^{m_{\mathcal{P}}-1} (k - m - i).$$

Since m and $m_{\mathcal{P}}$ do not depend on k, this concludes the proof of the claim. $\square$

Proof of Lemma 3.4.4: Let $I \in \mathit{inst}(S, \Sigma)$, $p \in \mathit{Pos}(I)$, and $\bar{a} \in \Omega(I, p)$. To prove this lemma it suffices to show that the following limit exists:

$$\lim_{k\to\infty} \frac{1}{\log k} \sum_{a \in [1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})}. \tag{A.1}$$

By Claims A.1.1 and A.1.2, there exists $k_0 > 0$ and polynomials $q_a(k)$, for every $a \in \mathit{adom}(I)$, and $q(k)$ such that for every $k > k_0$:

1. $|SAT^k_\Sigma(I_{(a,\bar{a})})| = q_a(k)$, for every $a \in \mathit{adom}(I)$;

2. $|SAT^k_\Sigma(I_{(a,\bar{a})})| = q(k)$, for every $a \in [1, k] - \mathit{adom}(I)$.

Let $n = |\mathit{adom}(I)|$ and $r(k) = (k - n)q(k) + \sum_{a \in \mathit{adom}(I)} q_a(k)$. Then (A.1) is equal to

$$\lim_{k\to\infty} \frac{1}{\log k} \Bigg[\sum_{a \in \mathit{adom}(I)} \left(\frac{q_a(k)}{r(k)} \log \frac{r(k)}{q_a(k)} \right) + (k - n) \frac{q(k)}{r(k)} \log \frac{r(k)}{q(k)} \Bigg]. \tag{A.2}$$

We first show that

$$\lim_{k\to\infty} \frac{1}{\log k}\Bigg[\sum_{a\in adom(I)} \frac{q_a(k)}{r(k)} \log \frac{r(k)}{q_a(k)}\Bigg] \;=\; 0. \tag{A.3}$$

Note that $degree(r) \geq degree(q_a)$ for every $a \in adom(I)$. If $degree(r) > degree(q_a)$, then clearly $\lim_{k\to\infty} \frac{q_a(k)}{r(k)} \log \frac{r(k)}{q_a(k)} = 0$. If $degree(r) = degree(q_a)$, then $\lim_{k\to\infty} \frac{q_a(k)}{r(k)} \log \frac{r(k)}{q_a(k)}$ exists and equals some positive constant c_a; hence $\lim_{k\to\infty} \frac{1}{\log k} \frac{q_a(k)}{r(k)} \log \frac{r(k)}{q_a(k)} = 0$. Thus, (A.3) holds and (A.2) equals

$$\lim_{k\to\infty} \Bigg[\frac{(k-n)}{\log k} \cdot \frac{q(k)}{r(k)} \cdot \log \frac{r(k)}{q(k)}\Bigg]. \tag{A.4}$$

By the definition of r, $degree(r) \geq degree(q) + 1$. A simple calculation shows that for $degree(r) = degree(q) + 1$, (A.4) equals some positive constant that depends on the coefficients of q and r, and for $degree(r) > degree(q) + 1$, (A.4) equals 0. Hence, the limit (A.2) always exists, which completes the proof. □

A.2 Proof of Lemma 3.5.2

Assume that

$$\lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \;\neq\; 0. \tag{A.5}$$

We will show that this limit must be 1.

First note that by (A.5), there exists $k_0 > 0$ such that for every $k \geq k_0$ and $a \in [1,k] - adom(I)$, $|SAT^k_\Sigma(I_{(a,\bar{a})})| \geq 1$. If this were not true, then by Claim A.1.1, for every $a \in \mathbb{N}^+ - adom(I)$, we would have $|SAT^k_\Sigma(I_{(a,\bar{a})})| = 0$ and, therefore, $\sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a|\bar{a})} \leq \log |adom(I)|$. We conclude that

$$\lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \;\leq\; \lim_{k\to\infty} \frac{\log |adom(I)|}{\log k} \;=\; 0,$$

which contradicts (A.5).

To prove the lemma we need to introduce an equivalence relation on the elements of $\bar{a}$ and prove some basic properties about it. Assume that $\|I\| = n, n > 0$. Let $k \geq k_0$ be such that $adom(I) \subsetneq [1,k]$. Given $a_i, a_j \in \bar{a}$, we say that a_i and a_j are *linked* in $(a, \bar{a})$, written as $a_i \sim a_j$, if for every substitution $\sigma : \bar{a} \to [1,k]$ such that $\sigma(I_{(a,\bar{a})}) \models \Sigma$, it is the case that $\sigma(a_i) = \sigma(a_j)$. Observe that if a_i, a_j are constants, then $a_i \sim a_j$ iff $a_i = a_j$. It is easy to see that $\sim$ is an equivalence relation on $\bar{a}$. We say that $a_i \in \bar{a}$ is *determined in* $(a, \bar{a})$ if for every pair of substitutions $\sigma_1, \sigma_2 : \bar{a} \to [1, k]$ such that $\sigma_1(I_{(a,\bar{a})}) \models \Sigma$ and $\sigma_2(I_{(a,\bar{a})}) \models \Sigma$, it is the case that $\sigma_1(a_i) = \sigma_2(a_i)$. Notice that if a_i is a constant, then a_i is determined in $(a, \bar{a})$. Furthermore, observe that if $a_i \sim a_j$ and a_i is determined in $(a, \bar{a})$, then a_j is determined in $(a, \bar{a})$. Thus, we can extend the definition for equivalence classes: $[a_i]_\sim$ is determined in $(a, \bar{a})$ if a_i is determined in $(a, \bar{a})$. We define $undet(a, \bar{a})$ as the set of all undetermined equivalence classes of $\sim$:

$$undet(a, \bar{a}) \quad = \quad \{[a_i]_\sim \mid a_i \in \bar{a} \text{ and } [a_i]_\sim \text{ is not determined}\}.$$

Claim A.2.1

1) *For every $a \in adom(I)$ and $b \in [1, k] - adom(I)$, if there exists a substitution $\sigma : \bar{a} \to [1, k]$ such that $\sigma(I_{(b,\bar{a})}) \models \Sigma$, then $|undet(b, \bar{a})| \geq |undet(a, \bar{a})|$.*

2) *For every $a, b \in [1, k] - adom(I)$, $undet(b, \bar{a}) = undet(a, \bar{a})$.*

PROOF: 1) Let $a \in adom(I)$ and $b \in [1, k] - adom(I)$. Assume that there exists a substitution $\sigma : \bar{a} \to [1, k]$ such that $\sigma(I_{(b,\bar{a})}) \models \Sigma$. It is easy to see that for every $a_i, a_j \in \bar{a}$, if a_i is determined in $(b, \bar{a})$, then a_i is determined in $(a, \bar{a})$, and if a_i, a_j are linked in $(b, \bar{a})$, then a_i, a_j are linked in $(a, \bar{a})$. Thus, $|undet(b, \bar{a})| \geq |undet(a, \bar{a})|$.

2) Trivial, by Claim A.1.1. □

Claim A.2.2 *Let $a \in [1, k] - adom(I)$. If $k > 2n$, then $|SAT^k_\Sigma(I_{(a,\bar{a})})| \geq (k - 2n)^{|undet(a,\bar{a})|}$.*

PROOF: To prove this claim, we consider two cases. First assume that $\bar{a}$ does not contain any variable. Then $|undet(a, \bar{a})| = 0$ and we have to prove that $|SAT^k_\Sigma(I_{(a,\bar{a})})| \geq 1$. For that, it suffices to show that $I_{(a,\bar{a})} \models \Sigma$. Towards a contradiction, assume that $I_{(a,\bar{a})} \not\models \Sigma$. Then by Claim A.1.1, $|SAT^k_\Sigma(I_{(b,\bar{a})})| = 0$, for every $b \in \mathbb{N}^+ - adom(I)$, which contradicts the existence of k_0.

Second assume that $\bar{a}$ contains at least one variable. Let $\sigma_0 : \bar{a} \to [1, k]$ be a substitution such that $\sigma_0(I_{(a,\bar{a})}) \models \Sigma$ (such a substitution exists by assumption (A.5)). Let $\sigma : \bar{a} \to [1, k]$ be a substitution such that: (a) σ and σ_0 coincide in determined equivalence classes; (b) for every undetermined class $[a_i]_\sim$, σ assigns the same value in $[1, k] - (adom(I) \cup \{a\})$ to each element in this class; (c) for every pair of distinct undetermined classes $[a_i]_\sim$, $[a_j]_\sim$, $\sigma(a_i) \neq \sigma(a_j)$. Notice that such a function exists since $k > 2n$. Given that $\sigma_0(I_{(a,\bar{a})}) \models \Sigma$, we have $\sigma(I_{(a,\bar{a})}) \models \Sigma$. Thus, $|SAT^k_\Sigma(I_{(a,\bar{a})})|$ is greater than or equal to the number of substitutions with domain $\bar{a}$ and range contained in $[1,k]$ satisfying conditions (a), (b) and (c). Therefore, $|SAT^k_\Sigma(I_{(a,\bar{a})})| \geq (k-(n+1))(k-(n+2))\cdots(k-(n+|undet(a,\bar{a})|)) \geq (k-2n)^{|undet(a,\bar{a})|}$. This proves the claim. $\square$

We will use this claim to prove that $\lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a|\bar{a})} = 1$. Let $k \geq k_0$ be such that $adom(I) \subseteq [1,k]$ and $k > 2n$. By Claim A.2.2, for every $a \in [1,k] - adom(I)$, $|SAT^k_\Sigma(I_{(a,\bar{a})})| \geq (k-2n)^{|undet(a,\bar{a})|}$. Furthermore, by Claim A.2.1, for every $a \in [1,k] - adom(I)$:

$$\sum_{b\in[1,k]} |SAT^k_\Sigma(I_{(b,\bar{a})})| \;\leq\; \sum_{b\in[1,k]} k^{|undet(b,\bar{a})|} \;\leq\; k^{|undet(a,\bar{a})|+1}$$

Thus, for every $a \in [1,k] - adom(I)$:

$$P(a \mid \bar{a}) \;\geq\; \frac{(k-2n)^{|undet(a,\bar{a})|}}{k^{|undet(a,\bar{a})|+1}} \;=\; \frac{1}{k}\,(1-\frac{2n}{k})^{|undet(a,\bar{a})|}. \tag{A.6}$$

By Claim A.1.1, for every $a, b \in [1,k] - adom(I)$, $P(a \mid \bar{a}) = P(b \mid \bar{a})$ and, therefore,

$$P(a \mid \bar{a}) \;\leq\; \frac{1}{k - |adom(I)|} \;\leq\; \frac{1}{k-n}. \tag{A.7}$$

Therefore, using (A.6) and (A.7) we conclude that:

$$\begin{aligned}\sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \;&\geq\; \sum_{a\in[1,k]-adom(I)} \frac{1}{k}\,(1-\frac{2n}{k})^{|undet(b,\bar{a})|} \log(k-n)\\ &\geq\; \log(k-n)\,(1-\frac{n}{k})(1-\frac{2n}{k})^{|undet(b,\bar{a})|},\end{aligned}$$

where b is an arbitrary element in $[1,k] - adom(I)$. Thus,

$$\frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \;\geq\; \frac{\log(k-n)}{\log k}\,(1-\frac{n}{k})(1-\frac{2n}{k})^{|undet(b,\bar{a})|}.$$

It is straightforward to prove that $\lim_{k\to\infty}[\frac{\log(k-n)}{\log k}\,(1-\frac{n}{k})(1-\frac{2n}{k})^{|undet(b,\bar{a})|}] = 1$. Thus,

$$\lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} \geq 1$$

and, therefore,

$$\lim_{k\to\infty} \frac{1}{\log k} \sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a \mid \bar{a})} = 1,$$

since $\sum_{a\in[1,k]} P(a \mid \bar{a}) \log \frac{1}{P(a|\bar{a})} \leq \log k$. This completes the proof of Lemma 3.5.2.

Appendix B

Proofs from Chapter 5

In this chapter we use the following notations. Referring to an XML tree $T = (V,\ lab,\ ele,\ att,\ root)$ conforming to a DTD $D = (E,\ A,\ P,\ R,\ r)$, for every element type $\tau \in E$ and $@l \in R(\tau)$, $values(\tau.@l)$ denotes $\{x.@l \mid x \in ext(\tau)\}$, the set of all the $@l$-attribute values of τ-nodes in T. We write $|S|$ for the cardinality of a set S. Given a DTD D and a set Σ of constraints, we also use $|D|$ and $|\Sigma|$ to denote their sizes, respectively. Finally, we write $T \models (D, \Sigma)$ instead of $T \models D$ and $T \models \Sigma$.

B.1 Proof of Theorem 5.3.1

The proof consists of two PTIME reductions, one for each direction.

a) A reduction from $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ *to PDE.* We first define a class of simplified DTDs called *narrow DTDs*, and we explain how to reduce the consistency problem for $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over arbitrary DTDs to that over narrow DTDs. Then we show how to encode the consistency problem for narrow DTDs and $\mathcal{AC}^{*,1}_{PK,FK}$-constraints by a prequadratic Diophantine system.

We start by explaining the process of narrowing the DTDs. Intuitively, we replace long "horizontal" regular expressions in $P(\tau)$ by shorter ones. Formally, consider a DTD $D = (E,\ A,\ P,\ R,\ r)$. D is basically an extended regular grammar (cf. [CGL99, Nev99]); for each $\tau \in E$, $P(\tau)$ is a regular expression α and, thus, $\tau \to \alpha$ can be viewed as the production rule for τ. We rewrite the regular expression by introducing a set N of new element types (nonterminals) such that the production rules of the new DTD have one of the following forms:

$$\tau \to \tau_1,\ \tau_2 \qquad \tau \to \tau_1 \mid \tau_2 \qquad \tau \to \tau_1^* \qquad \tau \to \tau' \qquad \tau \to \ \mathtt{S} \qquad \tau \to \epsilon$$

where τ, τ_1, τ_2 are element types in $E \cup N$, $\tau' \in E$, $\mathtt{S}$ is the string type and ϵ denotes the empty word. More specifically, we conduct the following "narrowing" process on the production rule $\tau \to \alpha$:

- If $\alpha = (\alpha_1, \alpha_2)$, then we introduce two new element types τ_1, τ_2 and replace $\tau \to \alpha$ with a new rule $\tau \to \tau_1, \tau_2$. We proceed to process $\tau_1 \to \alpha_1$ and $\tau_2 \to \alpha_2$ in the same way.

- If $\alpha = (\alpha_1|\alpha_2)$, then we introduce two new element types τ_1, τ_2 and replace $\tau \to \alpha$ with a new rule $\tau \to \tau_1 \mid \tau_2$. We proceed to process $\tau_1 \to \alpha_1$ and $\tau_2 \to \alpha_2$ in the same way.

- If $\alpha = \alpha_1^*$, then we introduce a new element type τ_1 and replace $\tau \to \alpha$ with $\tau \to \tau_1^*$. We proceed to process $\tau_1 \to \alpha_1$ in the same way.

- If α is one of $\tau' \in E$, $\mathtt{S}$ or ϵ, then the rule for τ remains unchanged.

We refer to the set of new element types introduced when processing $\tau \to P(\tau)$ as N_τ and the set of production rules generated/revised as P_τ. Observe that $N_\tau \cap E = \emptyset$ for any $\tau \in E$. We define a new DTD $D_N = (E_N,\ A,\ P_N,\ R_N,\ r)$, referred to as the *narrowed DTD of D* (or just a narrow DTD if D is clear from the context), where

- $E_N = E \ \cup \ \bigcup_{\tau \in E} N_\tau$, i.e., all element types of E and new element types introduced in the narrowing process;

- $P_N = \bigcup_{\tau \in E} P_\tau$, i.e., production rules generated/revised in the narrowing process;

- $R_N(\tau) = R(\tau)$ for each $\tau \in E$, and $R_N(\tau) = \emptyset$ for each $\tau \in E_N - E$.

Note that the root element type r and the set A of attributes remain unchanged. Moreover, elements of any type in $E_N - E$ do not have any attribute. The only kind of P_N production rules whose right-hand side contains element type of E are of the form $\tau \to \tau'$, where $\tau' \in E$. It is easy to see that D_N is computable in polynomial time.

Obviously, any set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D is also a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over the narrow DTD D_N of D. The next lemma establishes the connection between D and D_N, which allows us to consider only narrow DTDs from now on.

Lemma B.1.1 *Let D be a DTD, D_N the narrowed DTD of D and Σ a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D. Then there exists an XML tree T_1 such that $T_1 \models (D, \Sigma)$ iff there exists an XML tree T_2 such that $T_2 \models (D_N, \Sigma)$.*

Proof: Given an element type τ and a sequence of attributes $@l_1, \ldots, @l_n \in R(\tau)$, define $values(\tau[@l_1, \ldots, @l_n])$ as $\{(x.@l_1, \ldots, x.@l_n) \mid x \in ext(\tau)\}$.

To prove the lemma, it suffices to show the following:

Claim: Given any XML tree $T_1 \models D$ one can construct an XML tree T_2 by modifying T_1 such that $T_2 \models D_N$, and vice versa. Furthermore, for every element type τ in D and $@l_1, \ldots, @l_n \in R(\tau)$, $|ext(\tau)|$ in T_2 equals $|ext(\tau)|$ in T_1, and $values(\tau[@l_1, \ldots, @l_n])$ in T_2 equals $values(\tau[@l_1, \ldots, @l_n])$ in T_1.

For if the claim holds, we can show the lemma as follows. Assume that there exists an XML tree T_1 such that $T_1 \models D$ and $T_1 \models \Sigma$. By the claim, there is T_2 such that $T_2 \models D_N$. Suppose, by contradiction, there is $\varphi \in \Sigma$ such that $T_2 \not\models \varphi$. (1) If φ is a key $\tau[@l_1, \ldots, @l_n] \to \tau$, then there exist two distinct nodes $x, y \in ext(\tau)$ in T_2 such that $x.@l_i = y.@l_i$ for every $i \in [1, n]$. In other words, $|values(\tau[@l_1, \ldots, @l_n])| < |ext(\tau)|$ in T_2. Since $T_1 \models \varphi$, it must be the case that $|values(\tau[@l_1, \ldots, @l_n])| = |ext(\tau)|$ in T_1 because the tuple $(x.@l_1, \ldots, x.@l_n)$ of each $x \in ext(\tau)$ uniquely identifies x among $ext(\tau)$. This contradicts the claim that $|ext(\tau)|$ in T_2 equals $|ext(\tau)|$ in T_1 and $values(\tau[@l_1, \ldots, @l_n])$ in T_2 equals $values(\tau[@l_1, \ldots, @l_n])$ in T_1. (2) If φ is a unary foreign key: $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$, then either $T_2 \not\models \tau_2.@l_2 \to \tau_2$ or there is $x \in ext(\tau_1)$ in T_2 such that for all $y \in ext(\tau_2)$ in T_2, $x.@l_1 \neq y.@l_2$. In the first case, we reach a contradiction as in (1). In the second case, we have $x.@l_1 \notin values(\tau_2.@l_2)$ in T_2. By the claim, $x.@l_1 \in values(\tau_1.@l_1)$ in T_1. Since $T_1 \models \varphi$, $x.@l_1 \in values(\tau_2.@l_2)$ in T_1. Again by the claim, we have $x.@l_1 \in values(\tau_2.@l_2)$ in T_2, which contradicts the assumption. The proof for the other direction is similar.

We next verify the claim. Given an XML tree $T_1 = (V_1, lab_1, ele_1, att, root)$ such that $T_1 \models D$, we construct an XML tree T_2 by modifying T_1 such that $T_2 \models D_N$. Consider a τ-element v in T_1. Let $ele_1(v) = [v_1, ..., v_n]$ and $w = lab_1(v_1) \ldots lab_1(v_n)$. Recall N_τ and P_τ, the set of nonterminals and the set of production rules generated when narrowing $\tau \to P(\tau)$. Let Q_τ be the set of E symbols that appear in P_τ plus $\mathtt{S}$. We can view $G = (Q_\tau, N_\tau \cup \{\tau\}, P_\tau, \tau)$ as an extended context free grammar, where Q_τ is the set of terminals, $N_\tau \cup \{\tau\}$ the set of nonterminals, P_τ the set of production rules and τ the start symbol[1]. Since $T_1 \models D$, we have $w \in P(\tau)$. By a straightforward induction on the structure of $P_N(\tau)$ it can be verified that w is in the language defined by G. Thus there is a parse tree $T(w)$ w.r.t. the grammar G for w, and w is the frontier (the list of leaves from left to right) of $T(w)$. Without loss of generality, assume that the root of $T(w)$ is v, and the leaves are $v_1, \ldots, v_n$. Observe that the internal nodes of $T(w)$ are labeled with element types in N_τ except that the root v is labeled τ. Intuitively, we construct T_2 by replacing each element v in T_1 by such a parse tree. More specifically, let $T_2 = (V_2, \mathit{lab}_2, \mathit{ele}_2, \mathit{att}, \mathit{root})$. Here V_2 consists of nodes in V_1 and the internal nodes introduced in the parse trees. For each x in V_2, let $\mathit{lab}_2(x) = \mathit{lab}_1(x)$ if $x \in V_1$, and otherwise let $\mathit{lab}_2(x)$ be the node label of x in the parse tree where x belongs. Note that nodes in $V_2 - V_1$ are elements of some type in $E_N - E$. For every $x \in V_1$, let $\mathit{ele}_2(x)$ be the list of its children in the parse tree having x as root. For every $x \in V_2 - V_1$, let $\mathit{ele}_2(x)$ be the list of its children in the parse tree containing x. Note that att and root remain unchanged. By the construction of T_2 it can be verified that $T_2 \models D_N$; and moreover, for every element type τ in D and $@l_1, \ldots, @l_n \in R(\tau)$, $|\mathit{ext}(\tau)|$ in T_2 equals $|\mathit{ext}(\tau)|$ in T_1 and $\mathit{values}(\tau[@l_1, \ldots, @l_n])$ in T_2 equals $\mathit{values}(\tau[@l_1, \ldots, @l_n])$ in T_1 because, among other things, (1) none of the new nodes, i.e., nodes in $V_2 - V_1$, is labeled with an E-type; (2) no new attributes are defined; and (3) attribute function att is unchanged.

Conversely, assume that there is $T_2 = (V_2, \mathit{lab}_2, \mathit{ele}_2, \mathit{att}, \mathit{root})$ such that $T_2 \models D_N$. We construct an XML tree T_1 by modifying T_2 such that $T_1 \models D$. For every node $v \in V_2$ with $\mathit{lab}(v) = \tau$ and $\tau \in E_N - E$, we substitute v in $\mathit{ele}_2(v')$ by the children of v, where v' is the parent of v. In addition, we remove v from V_2, $\mathit{lab}_2(v)$ from lab_2, and $\mathit{ele}_2(v)$ from ele_2. Observe that by the definition of D_N, no attributes are defined for elements of any type in $E_N - E$. We repeat the process until there is no node labeled with element type in $E_N - E$. Now let $T_1 = (V_1, \mathit{lab}_1, \mathit{ele}_1, \mathit{att}, \mathit{root})$, where V_1, lab_1 and ele_1 are V_2, lab_2 and ele_2 at the end of the process, respectively. Notice that att and root remain unchanged. By the definition of T_1 it can be verified that $T_1 \models D$; and in addition, for every element type τ in D and $@l_1, \ldots, @l_n \in R(\tau)$, $|\mathit{ext}(\tau)|$ in T_2 equals $|\mathit{ext}(\tau)|$ in T_1 and $\mathit{values}(\tau[@l_1, \ldots, @l_n])$ in T_2 equals $\mathit{values}(\tau[@l_1, \ldots, @l_n])$ in T_1 because, among other things, none of the nodes removed is labeled with a type of E and the attribute function att is unchanged. □

By Lemma B.1.1, in the rest of this proof we consider only narrow DTDs. Next we

[1]If τ is in $P(\tau)$, i.e., if τ is recursively defined, we need to rename τ in Q_τ to ensure that Q_τ and $N_\tau \cup \{\tau\}$ are disjoint. It is straightforward to handle that case.

show how to encode $\mathcal{AC}^{*,1}_{PK,FK}$-constraints by a prequadratic Diophantine system. Let $D = (E, A, P, R, r)$ be a narrow DTD and Σ be a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints, i.e., primary $\mathcal{AC}^{*,1}_{K,FK}$-constraints. We encode Σ with a set C_Σ of integer constraints, referred to as *the cardinality constraints determined by* Σ. For every $\varphi \in \Sigma$,

- if φ is a key constraint $\tau[@l_1, \ldots, @l_k] \to \tau$, then C_Σ contains $|ext(\tau)| \leq |values(\tau.@l_1)| \cdot \ldots \cdot |values(\tau.@l_k)|$;

- if φ is a unary foreign key $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$, then C_Σ contains $|values(\tau_1.@l_1)| \leq |values(\tau_2.@l_2)|$ and $|ext(\tau_2)| \leq |values(\tau_2.@l_2)|$;

- furthermore, for any $\tau \in E$, if $R(\tau) = \emptyset$, then $0 \leq |ext(\tau)|$ is in C_Σ. Otherwise, for every $@l \in R(\tau)$, $|values(\tau.@l)| \leq |ext(\tau)|$ and $0 \leq |values(\tau.@l)|$ are in C_Σ.

Observe that for a unary key $\tau.@l \to \tau$ we have both $|values(\tau.@l)| \leq |ext(\tau)|$ and $|ext(\tau)| \leq |values(\tau.@l)|$ in C_Σ. Thus C_Σ assures $|ext(\tau)| = |values(\tau.@l)|$.

We write $T \models C_\Sigma$ if T satisfies all the constraints of C_Σ, and we write $T \models (D, C_\Sigma)$ if T conforms to a narrow DTD D and satisfies C_Σ. Note that C_Σ is equivalent (in fact, can be converted in polynomial time) to a prequadratic Diophantine system since $x \leq x_1 \cdot \ldots \cdot x_k$ can be written as constraints of the form $x \leq y \cdot z$ by introducing $k-2$ fresh variables, e.g., $x \leq x_1 \cdot x_2 \cdot x_3 \cdot x_4$ is equivalent to $x \leq x_1 \cdot z_1$, $z_1 \leq x_2 \cdot z_2$ and $z_2 \leq x_3 \cdot x_4$ (in the sense that the former is satisfiable iff the latter is). Thus, without loss of generality, assume that C_Σ consists of linear and prequadratic integer constraints only. It should be noted that C_Σ can be computed in time polynomial in the size of Σ and D. The lemma below shows that C_Σ characterizes the consistency of Σ if keys in Σ are primary.

Lemma B.1.2 *Let D be a narrow DTD and Σ a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D. Then every XML tree conforming to D and satisfying Σ also satisfies C_Σ. In addition, if there exists an XML tree T_2 such that $T_2 \models (D, C_\Sigma)$, then there exists an XML tree T_1 such that $T_1 \models (D, \Sigma)$.*

PROOF: It is easy to see that for every XML tree T_1 that satisfies Σ, it must be the case that $T_1 \models C_\Sigma$.

Conversely, we show that if there exists an XML tree $T_2 = (V, lab, ele, att_2, root)$ such that $T_2 \models (D, C_\Sigma)$, then we can construct an XML tree $T_1 = (V, lab, ele, att_1, root)$ such that $T_1 \models (D, \Sigma)$. We construct T_1 from T_2 by modifying the function att_2 while leaving

V, lab, ele and $root$ unchanged. More specifically, let $S = \{\tau.@l \mid \tau \in E, @l \in R(\tau)\}$. To define the new function, denoted by att_1, we first associate a set of string values with each $\tau.@l$ in S. Let N be the maximum cardinality of $values(\tau.@l)$ in T_2, i.e., $N \geq |values(\tau.@l)|$ in T_2 for all $\tau.@l \in S$. Let $V_S = \{a_i \mid i \in [1, N]\}$ be a set of distinct string values. For each $\tau.@l \in S$, let $V_{\tau.@l} = \{a_i \mid i \in [1, |values(\tau.@l)|]\}$, and for each $x \in ext(\tau)$, let $att(x, @l)$ be a string value in $V_{\tau.@l}$ such that in T_1, $values(\tau.@l) = V_{\tau.@l}$. In addition, for each key $\varphi = \tau[@l_1, \ldots, @l_k] \to \tau$ in Σ, let $x[@l_1, \ldots, @l_k]$ be a distinct list of string values from $V_{\tau.@l_1} \times \ldots \times V_{\tau.@l_k}$. This is possible because by the definition of T_1, (1) $ext(\tau)$ in T_1 equals $ext(\tau)$ in T_2; (2) $|values(\tau.@l)|$ in T_1 equals $|values(\tau.@l)|$ in T_2; (3) $T_2 \models C_\Sigma$ and $|ext(\tau)| \leq |values(\tau.@l_1)| \cdot \ldots \cdot |values(\tau.@l_k)|$ is in C_Σ; and (4) since φ is the only key defined for τ-elements, the population of the attributes $@l_1, \ldots, @l_k$ of x is independent of the population of any other attributes of x. It should be noted that it may be the case that $V_{\tau_1.@l_1} \subseteq V_{\tau_2.@l_2}$ even if Σ does not imply $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$. This does not lose generality as we do not intend to capture negation of foreign keys. We next show that T_1 is indeed what we want.

It is easy to verify that $T_1 \models D$ given the construction of T_1 from T_2 and the assumption that $T_2 \models D$. To show that $T_1 \models \Sigma$, we consider $\varphi \in \Sigma$ in the following cases. (1) If φ is a key $\tau[@l_1, \ldots, @l_k] \to \tau$, it is immediate from the definition of T_1 that $T_1 \models \varphi$ since for any $x \in ext(\tau)$, $x[@l_1, \ldots, @l_k]$ is a distinct list of string values from $V_{\tau.@l_1} \times \ldots \times V_{\tau.@l_k}$. (2) If φ is $\tau_1.@l_1 \subseteq_{FK} \tau_2.@l_2$, then $T_2 \models |values(\tau_1.@l_1)| \leq |values(\tau_2.@l_2)|$ by $T_2 \models C_\Sigma$. By the definition of att_1, for $i = 1, 2$, $V_{\tau_i.@l_i} = \{a_i \mid i \in [1, |values(\tau_i.@l_i)|]\}$ and in T_1, $values(\tau_i.@l_i) = V_{\tau_i.@l_i}$. Thus $values(\tau_1.@l_1) \subseteq values(\tau_2.@l_2)$ in T_1. Furthermore, given that $|ext(\tau_2)| \leq |values(\tau_2.@l_2)|$ and $|values(\tau_2.@l_2)| \leq |ext(\tau_2)|$ are both in C_Σ, $T_2 \models C_\Sigma$, $|ext(\tau_2)|$ in T_2 is equal to $|ext(\tau_2)|$ in T_1 and $|values(\tau_2.@l_2)|$ in T_2 is equal to $|values(\tau_2.@l_2)|$ in T_1, we conclude that $|ext(\tau_2)|$ is equal to $|values(\tau_2.@l_2)|$ in T_1 and, hence, $T_1 \models \tau_2.@l_2 \to \tau_2$ since each $x \in ext(\tau_2)$ in T_1 has a distinct $@l_2$-attribute value and thus the value of its $@l_2$-attribute uniquely identifies x among nodes in $ext(\tau_2)$. Therefore, $T_1 \models \varphi$ and, thus, $T_1 \models (D, \Sigma)$. This concludes the proof of the lemma. $\square$

The above lemma takes care of coding the constraints; the next step is to code DTDs. For that, we use the technique developed in [FL02]: for each narrow DTD D, one can compute in polynomial time in the size of D a set Ψ_D of linear inequalities on nonnegative integers, referred to as *the set of cardinality constraints determined by* D, which includes $|ext(\tau)|$ as a variable for each element type τ in D, but it does not have $|values(\tau.@l)|$ as a variable for any attribute $@l$ of τ. Moreover, the following has been shown [FL02]: Ψ_D

has a nonnegative integer solution if and only if there exists an XML tree T conforming to D such that the cardinality of $ext(\tau)$ in T equals the value of the variable $|ext(\tau)|$ in the solution for each element type τ in D.

We now combine this coding with the coding for $\mathcal{AC}^{*,1}_{PK,FK}$-constraints. Given a narrow DTD D and a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D, we define *the set of cardinality constraints determined by D and Σ* to be

$$\Psi(D,\,\Sigma) \;=\; \Psi_D \cup C_\Sigma \cup \{(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0) \mid \tau \in E,\; @l \in R(\tau)\},$$

where C_Σ is the set of cardinality constraints determined by Σ, Ψ_D is the set of cardinality constraints determined by D, and constraints $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ are to ensure that every τ-element has an $@l$-attribute (note that $|values(\tau.@l)| \leq |ext(\tau)|$ is already in C_Σ). Constraints in $\Psi(D,\,\Sigma)$ are either linear integer constraints, or inequalities of the form $x \leq y \cdot z$, which come from C_Σ, or constraints of the form $x > 0 \rightarrow y > 0$. Note that if we leave out constraints of the form $x > 0 \rightarrow y > 0$, $\Psi(D,\,\Sigma)$ is a prequadratic Diophantine system. Also note that $\Psi(D,\,\Sigma)$ can be computed in polynomial time in the size of D and Σ.

We say that $\Psi(D,\,\Sigma)$ is *consistent* if and only if $\Psi(D,\,\Sigma)$ admits a nonnegative integer solution. That is, there is a nonnegative integer assignment to the variables in $\Psi(D,\,\Sigma)$ such that all the constraints in $\Psi(D,\,\Sigma)$ are satisfied.

Lemma B.1.3 *Let D be a narrow DTD and Σ a set of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D. Then $\Psi(D,\,\Sigma)$ is consistent if and only if there is an XML tree T such that $T \models (D, \Sigma)$.*

Proof: Suppose that there exists an XML tree T such that $T \models (D, \Sigma)$. Then there is a nonnegative integer solution to Ψ_D such that for each element type τ in D, the value of the variable $|ext(\tau)|$ equals the number of τ-elements in T [FL02]. By Lemma B.1.2 and $T \models \Sigma$, we have $T \models C_\Sigma$. We extend the solution of Ψ_D to be one to $\Psi(D,\,\Sigma)$ by letting the variable $|values(\tau.@l)|$ equal the number of distinct $@l$-attribute values of all τ-elements in T, for each element type τ and attribute $@l$ of τ in D. Since $T \models C_\Sigma$, this extended assignment satisfies all the constraints in C_Σ. In addition, if $|ext(\tau)| > 0$ then $|values(\tau.@l)| > 0$ since every τ-element in T has an $@l$-attribute. Hence the assignment is indeed a nonnegative solution to $\Psi(D,\,\Sigma)$ and, therefore, $\Psi(D,\,\Sigma)$ is consistent.

Conversely, suppose that $\Psi(D,\,\Sigma)$ admits a nonnegative integer solution. Then there exists an XML tree T such that $T \models D$ and moreover, for each element type τ in D, the cardinality of $ext(\tau)$ in T equals the value of the variable $|ext(\tau)|$ in the solution

[FL02]. We construct a new tree T' from T by modifying the definition of the function *att* such that in T', for each element type τ and attribute $@l$ of τ, the number of distinct $@l$-attribute values of all τ-elements equals the value of the variable $|values(\tau.@l)|$ in the solution. This is possible since $|values(\tau.@l)| \leq |ext(\tau)|$ and $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ are in $\Psi(D, \Sigma)$. The assignment is also a solution to C_Σ. Thus $T' \models D$ and $T' \models C_\Sigma$. Hence by Lemma B.1.2, there exists an XML tree T'' such that $T'' \models (D, \Sigma)$. This concludes the proof of the lemma. $\square$

We now conclude the proof of reduction from $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ to PDE. By Lemma B.1.1, given an arbitrary DTD D and a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints, one can compute a narrow DTD D_N such that (D, Σ) is consistent iff (D_N, Σ) is consistent. By Lemma B.1.3, (D_N, Σ) is consistent iff $\Psi(D_N, \Sigma)$ has a nonnegative integer solution. Such a solution requires $|values(\tau.@l)| > 0$ if $|ext(\tau)| > 0$. To ensure this, let $\Phi(D_N, \Sigma)$ be a system that includes all linear integer constraints and prequadratic constraints in $\Psi(D_N, \Sigma)$ and moreover, $|ext(\tau)| \leq |values(\tau.@l)| \cdot |ext(\tau)|$ for each $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ in $\Psi(D_N, \Sigma)$. Now $\Phi(D_N, \Sigma)$ is a prequadratic Diophantine system. In addition, $\Psi(D_N, \Sigma)$ has a nonnegative integer solution iff $\Phi(D_N, \Sigma)$ has a nonnegative integer solution. To see this, observe that for any nonnegative integer assignment to $|ext(\tau)|$ and $|values(\tau.@l)|$, $(|ext(\tau)| > 0) \rightarrow (|values(\tau.@l)| > 0)$ iff $|ext(\tau)| \leq |values(\tau.@l)| \cdot |ext(\tau)|$. Thus, (D, Σ) is consistent iff the prequadratic Diophantine system $\Phi(D_N, \Sigma)$ has a nonnegative integer solution. Note that D_N can be computed in polynomial time in the size of D, $\Psi(D_N, \Sigma)$ can be computed in polynomial time in the size of D_N and Σ, and $\Phi(D_N, \Sigma)$ can be computed in polynomial time in the size of $\Psi(D_N, \Sigma)$. Hence, it takes polynomial time to compute $\Phi(D_N, \Sigma)$ from D and Σ. Therefore, there is a PTIME reduction from $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ to PDE.

b) A reduction from PDE to $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$. We now move to the other direction. Given an instance of PDE, i.e., a system S consisting of a set S_L of linear equations/inequalities on integers and a set S_P of prequadratic constraints of the form $x \leq y \cdot z$, we define a DTD D and a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints such that S has a nonnegative solution iff there is an XML tree T satisfying Σ and conforming to D. We use $X = \{x_i \mid i \in [1, n]\}$ to denote the set of all the variables in S. Assume that $S_L = \{e_j \mid j \in [1, m]\}$ and e_j is of the form: $a^j_1\ x_1 + \ldots + a^j_n\ x_n + c_j \leq b^j_1\ x_1 + \ldots + b^j_n\ x_n + d_j$, where a^j_i $(i \in [1, n])$, b^j_i $(i \in [1, n])$, c_j and d_j are nonnegative integers[2]. Also, assume that $S_P = \{p_j \mid j \in [1, l]\}$, where p_j is a

[2]For example, we represent equation $-3x + 5y \leq -7$ as $0x + 5y + 7 \leq 3x + 0y + 0$.

prequadratic equation of the form $x \leq y \cdot z$. Then we define DTD $D = (E,\ A,\ P,\ R,\ r)$ as follows:

(1) For each variable x_i, we define an element type X_i. In addition, for each $p_s \in S_P$ of the form $x_i \leq x_j \cdot x_k$, we define an element type U_i^s. For each linear constraint e_j, we define distinct element types E_j, A_1^j, ..., A_n^j, C_j, F_j, B_1^j, ..., B_n^j, D_j. We use r to denote the root element type. That is,

$$\begin{aligned} E \ = \ & \{r\} \cup \{X_i \mid i \in [1,n]\} \cup \\ & \{E_j, A_1^j, \ldots, A_n^j, C_j, F_j, B_1^j, \ldots, B_n^j, D_j \mid j \in [1,m]\} \cup \{U_i^s \mid p_s = x_i \leq x_j \cdot x_k \in S_P\}. \end{aligned}$$

Intuitively, referring to an XML tree conforming to D, we use $|ext(X_i)|$ to code the value of the variable x_i in S. For every equation e_j, we use $|ext(A_1^j)|$, ..., $|ext(A_n^j)|$, $|ext(C_j)|$ to code the values of constants a_1^j, ..., a_n^j, c_j, $|ext(E_j)|$ to code the value of the expression $a_1^j x_1 + \cdots + a_n^j x_n + c_j$, $|ext(B_1^j)|$, ..., $|ext(B_n^j)|$, $|ext(D_j)|$ to code the values of constants b_1^j, ..., b_n^j, d_j and $|ext(F_j)|$ to code the value of the expression $b_1^j x_1 + \cdots + b_n^j x_n + d_j$, Furthermore, for each prequadratic equation $p_s = x_i \leq x_j \cdot x_k$ in S_P, we create a distinct copy U_i^s of X_i. The reason to use U_i^s instead of X_i is to ensure that the set Σ of $\mathcal{AC}^{*,1}_{K,FK}$-constraints defined below is primary.

(2) $A = \{@c,\ @d,\ @e\}$. Intuitively, we shall define $@e$ as a key and use $@c$ and $@d$ to code prequadratic constraint of the form $x \leq y \cdot z$.

(3) We define production rules as follows. For the root of the DTD:

$$\begin{aligned} P(r) \ = \ & (X_1, U_1^{s_{1,1}}, \ldots, U_1^{s_{1,j_1}})^*, \ldots, (X_n, U_n^{s_{n,1}}, \ldots, U_n^{s_{n,j_n}})^*, \\ & \underbrace{C_1, \ldots, C_1}_{c_1 \text{ times}}, \ldots, \underbrace{C_m, \ldots, C_m}_{c_m \text{ times}}, \underbrace{D_1, \ldots, D_1}_{d_1 \text{ times}}, \ldots, \underbrace{D_m, \ldots, D_m}_{d_m \text{ times}}, \end{aligned}$$

where $\{s_{i,1},\ \ldots,\ s_{i,j_i}\}$ ($i \in [1,n]$) is the set of indexes $\{s \mid p_s = x_i \leq x_j \cdot x_k \in S_P\}$. Furthermore, for every $i \in [1,n]$ and every $j \in [1,m]$:

$$\begin{aligned} P(A_i^j) &= E_j, \\ P(C_j) &= E_j, \\ P(B_i^j) &= F_j, \\ P(D_j) &= F_j, \\ P(X_i) &= \underbrace{A_i^1, \ldots, A_i^1}_{a_i^1 \text{ times}}, \ldots, \underbrace{A_i^m, \ldots, A_i^m}_{a_i^m \text{ times}}, \underbrace{B_i^1, \ldots, B_i^1}_{b_i^1 \text{ times}}, \ldots, \underbrace{B_i^m, \ldots, B_i^m}_{b_i^m \text{ times}}. \end{aligned}$$

Finally, for every $i \in [1,n]$ and every $s \in [1,l]$ such that $p_s = x_i \leq x_j \cdot x_k \in S_P$, $P(U_i^s) = \epsilon$.

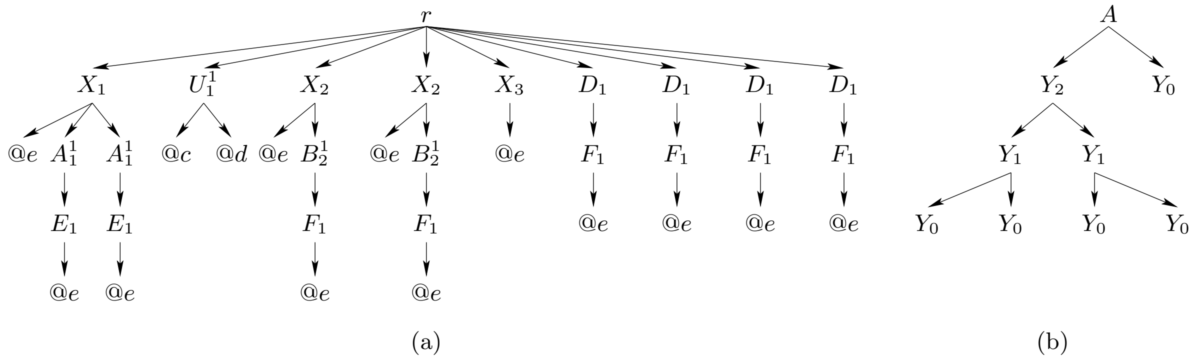


Figure B.1: Trees used in the proof of Theorem 5.3.1

(4) We define the attribute function R as follows: for every $j \in [1, m]$, $R(E_j) = R(F_j) = \{@e\}$. In addition, for every $i \in [1, n]$, $R(X_i) = \{@e\}$, and for every $s \in [1, l]$ such that $p_s = x_i \leq x_j \cdot x_k \in S_P$, $R(U_i^s) = \{@c, @d\}$. For all other element type τ, let $R(\tau)$ be empty.

For example, Figure B.1 (a) shows an XML tree conforming to the DTD constructed from the set of equations $S_L = \{2x_1 \leq x_2 + 4\}$ and $S_P = \{x_1 \leq x_2 \cdot x_3\}$. We note that this tree codes solution $x_1 = 1$, $x_2 = 2$, $x_3 = 1$ for this system of equations.

Given DTD D, we define a set Σ of $\mathcal{AC}^{*,1}_{PK,FK}$-constraints over D. For each $j \in [1, m]$, Σ includes keys $E_j.@e \to E_j$, $F_j.@e \to F_j$ and foreign key $E_j.@e \subseteq_{FK} F_j.@e$. Furthermore, for every $i, j, k \in [1, n]$ and $s \in [1, l]$ such that $p_s = x_i \leq x_j \cdot x_k \in S_P$, Σ includes the following constraints:

$$U_i^s[@c, @d] \to U_i^s, \qquad U_i^s.@c \subseteq_{FK} X_j.@e, \qquad U_i^s.@d \subseteq_{FK} X_k.@e.$$

Clearly, the set Σ is primary, i.e., for any element type τ there is at most one key defined. In fact, we use copies U_i^s of X_i just to ensure that Σ is primary.

We next show that the encoding is indeed a reduction from PDE to $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$. Suppose that S has a nonnegative solution. Then we construct an XML tree T conforming to D as shown in Figure B.1 (a). That is, for each $i \in [1, n]$ we let $|ext(X_i)|$ be the value of the variable x_i in the solution. We note that, by definition of D, this implies that for every $s \in [1, l]$ such that $p_s = x_i \leq x_j \cdot x_k \in S_P$, $|ext(U_i^s)|$ is also equal to the value of x_i in the solution. For every $i \in [1, n]$ and every X_i-element x in T, we let $x.@e$ be a distinct value such that in T, $|values(X_i.@e)| = |ext(X_i)|$. For every $j \in [1, m]$ and every E_j-element x in T, we let $x.@e$ be a distinct value such that in T, $|values(E_j.@e)| = |ext(E_j)|$. Likewise, we assign values to the $@e$-attribute of the nodes in $ext(F_j)$ in such a way that $|values(F_j.@e)| = |ext(F_j)|$ in T. Finally, for every $i, j, k \in [1, n]$ and $s \in [1, l]$ such that

$p_s = x_i \leq x_j \cdot x_k \in S_P$, and for every node x in T of type U_i^s, we let $x[@c, @d]$ be a distinct list of string values from $values(X_j.@e) \times values(X_k.@e)$. This is possible since $x_i \leq x_j \cdot x_k \in S_P$ and by definition of T, $|ext(U_i^s)| = |ext(X_i)| = x_i$, $|values(X_j.@e)| = |ext(X_j)| = x_j$ and $|values(X_k.@e)| = |ext(X_k)| = x_k$. Since T codes a solution of S, it is straightforward to prove that $T \models C_\Sigma$, the set of cardinality constraints determined by Σ. Thus, by Lemma B.1.2 we conclude that there exists an XML tree T' such that $T' \models (D, \Sigma)$ and, hence, (D, Σ) is consistent. Conversely, suppose that there exists an XML tree T such that $T \models (D, \Sigma)$. We construct a solution of S by letting variable x_i equal $|ext(X_i)|$ in T. By definitions of D and Σ, it is easy to verify that this is indeed a nonnegative integer solution for S. In particular, each $p_s = x_i \leq x_j \cdot x_k$ in S_P holds because $T \models (D, \Sigma)$ and, thus, $|ext(X_i)| = |ext(U_i^s)| \leq |values(U_i^s.@c)| \cdot |values(U_i^s.@d)| \leq |values(X_j.@e)| \cdot |values(X_k.@e)| \leq |ext(X_j)| \cdot |ext(X_k)|$.

We observe that the previous reduction is not polynomial since constants a_i^j, b_i^j ($i \in [1, n]$, $j \in [1, m]$) and c_j, d_j ($j \in [1, m]$) are coded in unary. To overcome this problem, next we show how code in a DTD the binary representation of a number. We introduce this coding separately to simplify the presentation of this proof.

Assume that $a = \sum_{i=0}^{k} a_i \cdot 2^i$, where each a_i ($i \in [0, k-1]$) is either 0 or 1 and $a_k = 1$, that is, the binary representation of a is $a_k a_{k-1} \cdots a_1 a_0$. To code a in a DTD we include element types $A, Y_0, \ldots, Y_k$ and we define P on these elements as follows:

$$P(Y_i) = \begin{cases} \epsilon & i = 0 \\ Y_{i-1}, Y_{i-1} & \text{Otherwise} \end{cases}$$

and $P(A) = Y_{i_1}, \ldots, Y_{i_l}$, where $i_l > \cdots > i_1 \geq 0$ and $\{i_1, \ldots, i_l\}$ is the set of indexes $\{j \in [0, k] \mid a_j = 1\}$. We note that the size of this set of rules is polynomial in the size of a. Furthermore, if an XML tree T conforms to this DTD, then $|ext(Y_0)| = a$ in T. For example, if $a = 5$, then $P(A) = Y_2, Y_0$, $P(Y_2) = Y_1, Y_1$, $P(Y_1) = Y_0, Y_0$ and $P(Y_0) = \epsilon$ and an XML tree conforming to these rules is of the form shown in Figure B.1 (b).

Thus, by using this coding in our original reduction of PDE to $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$ we can show that there is a PTIME reduction from PDE to $\mathsf{SAT}(\mathcal{AC}^{*,1}_{PK,FK})$. This completes the proof of Theorem 5.3.1.

B.2 Proof of Theorem 5.3.5

We reduce $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ to the existence of solution of an (almost) instance of linear integer programming, which happens to be of double-exponential size; hence the 2-NEXPTIME bound. For the lower bound, we encode the quantified boolean formula problem (QBF) as an instance of $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$.

Proof of a) The proof is a bit long, so we first give a rough outline. The idea is similar to the proof of the NP membership for $\mathsf{SAT}(\mathcal{AC}_{K,FK})$ [FL02]: we code both the DTD and the constraints with linear inequalities over integers. However, compared to the proof of [FL02], the current proof is considerably harder due to the following. First, regular expressions in DTDs ("horizontal" regular expressions) interact in a certain way with regular expressions in integrity constraints (those correspond to "vertical" paths through the trees). To eliminate this interaction, we first show how to reduce the problem to that over *narrow* DTDs, in which no wide horizontal regular expressions are allowed. The next problem is that regular expressions in constraints can interact with each other. Thus, to model them with linear inequalities, we extend the approach of [FL02] by taking into account all possible Boolean combinations of regular languages given by expressions used in constraints. The last problem is coding the DTDs in such a way that variables corresponding to each node have the information about the path leading to the node, and its relationship with the regular expressions used in constraints. For that, we adopt the technique of [AV99], and tag all the variables in the coding of DTDs with states of a certain automaton (the product automaton for all the automata corresponding to the regular expressions used in constraints).

Now it is time to fill in all the details. First, we need some additional notation. For every regular expression β and every attribute $@l$, we write $values(\beta.@l)$ to denote the set $\{y.@l \mid y \in nodes(\beta)$ and $y.@l$ is defined$\}$. Observe that for any $\tau \in E - \{r\}$, and $@l \in R(\tau)$, $values(r.\_^*.\tau.@l)$ corresponds to our original definition of $values(\tau.@l)$

We say that a DTD D is *one-attribute* if D contains only one attribute and no element type τ such that $P(\tau) = \mathtt{S}$. We start by showing that $\mathsf{SAT}(\mathcal{AC}^{reg}_{K,FK})$ can be reduced to the consistency problem for regular expression constraints over one-attribute DTDs. Let $D = (E, A, P, R, r)$ be a DTD and Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D. First, define DTD $D_U = (E_U, A_U, P_U, R_U, r)$ as follows. For every $\tau \in E$ and $@l \in R(\tau)$, assume that $\tau_{@l}$ is a fresh element type symbol. Then define E_U as $E \cup \{\tau_{@l} \mid \tau \in E$ and $@l \in R(\tau)\}$ and $A_U = \{@e\}$, where $@e$ is a fresh attribute symbol. Furthermore, define functions P_U

and R_U as:

- For every $\tau \in E$ such that $P(\tau) = \mathtt{S}$, if $R(\tau) = \{@l_1, \ldots, @l_n\}$, where $n \geq 0$, then $P_U(\tau) = \tau_{@l_1}, \ldots, \tau_{@l_n}$ and $R_U(\tau) = \emptyset$.
- For every $\tau \in E$ such that $P(\tau)$ is a regular expression over E, if $R(\tau) = \{@l_1, \ldots, @l_n\}$, where $n \geq 0$, then $P_U(\tau) = P(\tau), \tau_{@l_1}, \ldots, \tau_{@l_n}$ and $R_U(\tau) = \emptyset$.
- For every $\tau \in E$ and $@l \in R(\tau)$, $P_U(\tau_{@l}) = \epsilon$ and $R_U(\tau_{@l}) = \{@e\}$.

We note that if $P(\tau) = \mathtt{S}$ and $R(\tau) = \emptyset$, then $P_U(\tau) = \epsilon$.

Second, define set Σ_U of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D_U as follows. For every key constraint $\beta.\tau.@l \to \beta.\tau$ in Σ, $\beta.\tau.\tau_{@l}.@e \to \beta.\tau.\tau_{@l}$ is in Σ_U, and for every foreign key constraint $\beta.\tau.@l \subseteq_{FK} \beta'.\tau'.@l'$ in Σ, $\beta.\tau.\tau_{@l}.@e \subseteq_{FK} \beta'.\tau'.\tau'_{@l'}.@e$ is in Σ_U.

Lemma B.2.1 *Let D be a DTD, Σ be a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, and D_U, Σ_U be as defined above. Then there exists an XML tree T_1 such that $T_1 \models (D, \Sigma)$ iff there exists an XML tree T_2 such that $T_2 \models (D_U, \Sigma_U)$.*

Proof: ($\Rightarrow$) Let $T_1 = (V_1,\ lab_1,\ ele_1,\ att_1,\ root)$ be an XML tree such that $T_1 \models (D, \Sigma)$. We define an XML tree T_2 from T_1 such that $T_2 \models (D_U, \Sigma_U)$. More specifically, $T_2 = (V_2,\ lab_2,\ ele_2,\ att_2,\ root)$, where V_2, lab_2, ele_2 and att_2 are defined as follows. Let v be a node in T_1 such that $lab_1(v) = \tau \in E$ and $R(\tau) = \{@l_1, \ldots, @l_k\}$. Then V_2 contains node v and fresh nodes $v_{@l_1}, \ldots, v_{@l_k}$ such that $lab_2(v) = \tau$ and $lab_2(v_{@l_i}) = \tau_{@l_i}$, for every $i \in [1, k]$. Furthermore, if $ele_1(v) = [s]$, where $s \in Str$, then $ele_2(v) = [v_{@l_1}, \ldots, v_{@l_k}]$. Otherwise, $ele_1(v) = [v_1, \ldots, v_n]$, where $n \geq 0$ and each v_i is an element node, and $ele_2(v) = [v_1, \ldots, v_n, v_{@l_1}, \ldots, v_{@l_k}]$. Finally, $att_2(v, @e)$ is not defined and $att_2(v_{@l_i}, @e) = att_1(v, @l_i)$, for every $i \in [1, k]$. Next we show that $T_2 \models (D_U, \Sigma_U)$.

By definition of D_U and given that $T_1 \models D$, it is easy to see that $T_2 \models D_U$. Assume that $T_2 \not\models \Sigma_U$. Then there exists $\varphi \in \Sigma_U$ such that $T_2 \not\models \varphi$. (1) If φ is a key $\beta.\tau.\tau_{@l}.@e \to \beta.\tau.\tau_{@l}$, then there exists distinct $v_1, v_2 \in nodes(\beta.\tau.\tau_{@l})$ in T_2 such that $att_2(v_1, @e) = att_2(v_2, @e)$. Let u_1, u_2 be the parents of v_1, v_2 in T_2, respectively. By definition of D_U and given that $v_1 \neq v_2$, we have that $u_1 \neq u_2$. Thus, by definition of T_2, u_1 and u_2 are nodes in T_1 such that $u_1, u_2 \in nodes(\beta.\tau)$ and $att_1(u_1, @l) = att_1(u_2, @l) = att_2(v_1, @e)$. Therefore, $T_1 \not\models \beta.\tau.@l \to \beta.\tau$, which contradicts the fact that $T_1 \models \Sigma$. (2) If φ is a foreign key $\beta.\tau.\tau_{@l}.@e \subseteq_{FK} \beta'.\tau'.\tau'_{@l'}.@e$, then $T_2 \not\models \beta'.\tau'.\tau'_{@l'}.@e \to \beta'.\tau'.\tau'_{@l'}$ or there exists $v \in nodes(\beta.\tau.\tau_{@l})$ such that $att_2(v, @e) \notin values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2. In the former case, we reach a contradiction as in (1). In the latter case, assume that u is the parent of v in T_2. By definition of T_2, we have that u is a node in T_1 such that $u \in nodes(\beta.\tau)$ and $att_1(u, @l) = att_2(v, @e)$. Thus, given that $values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2 is equal to $values(\beta'.\tau'.@l')$ in T_1, we conclude that $att_1(u, @l) \notin values(\beta'.\tau'.@l')$ in T_1. Therefore, $T_1 \not\models \beta.\tau.@l \subseteq_{FK} \beta'.\tau'.@l'$, which contradicts the fact that $T_1 \models \Sigma$.

($\Leftarrow$) Let $T_2 = (V_2, lab_2, ele_2, att_2, root)$ be an XML tree such that $T_2 \models (D_U, \Sigma_U)$. We define an XML tree T_1 from T_2 such that $T_1 \models (D, \Sigma)$. More specifically, $T_1 = (V_1, lab_1, ele_1, att_1, root)$, where V_1, lab_1, ele_1 and att_1 are defined as follows. Let v be a node in T_2 such that $lab_2(v) = \tau$, $\tau \in E$ and $R(\tau) = \{@l_1, \ldots, @l_k\}$. Then V_1 also contains node v with $lab_1(v) = \tau$. Furthermore, if $P(\tau) = \mathtt{S}$, then $ele_2(v) = [v_{@l_1}, \ldots, v_{@l_k}]$, where $lab(v_{@l_j}) = \tau_{@l_j}$ ($j \in [1, k]$), and we define $ele_1(v)$ as $[s]$, where s is an arbitrary element in Str, and we define $att_1(v, @l_i)$ as $att_2(v_{@l_i}, @e)$, for every $i \in [1, k]$. Otherwise, $P(\tau)$ is a regular expression over E and $ele_2(v) = [v_1, \ldots, v_n, v_{@l_1}, \ldots, v_{@l_k}]$, where $lab(v_i) \in E$ ($i \in [1, n]$) and $lab(v_{@l_j}) = \tau_{@l_j}$ ($j \in [1, k]$), and we define $ele_1(v)$ as $[v_1, \ldots, v_n]$ and $att_1(v, @l_i)$ as $att_2(v_{@l_i}, @e)$, for every $i \in [1, k]$. Next we show that $T_1 \models (D, \Sigma)$.

By definition of D_U and given that $T_2 \models D_U$, it is easy to see that $T_1 \models D$. Assume that $T_1 \not\models \Sigma$. Then there exists $\varphi \in \Sigma$ such that $T_1 \not\models \varphi$. (1) If φ is a key $\beta.\tau.@l \rightarrow \beta.\tau$, then there exists distinct $u_1, u_2 \in nodes(\beta.\tau)$ in T_1 such that $att_1(u_1, @l) = att_1(u_2, @l)$. By definition of T_1, u_1 and u_2 are also in $nodes(\beta.\tau)$ in T_2. Let v_1, v_2 be the children of u_1, u_2 in T_2 of type $\tau_{@l}$, respectively. Given that $u_1 \neq u_2$, we have that $v_1 \neq v_2$. Thus, by definition of T_1, v_1 and v_2 are nodes in T_2 such that $v_1, v_2 \in nodes(\beta.\tau.\tau_{@l})$ and $att_2(v_1, @e) = att_2(v_2, @e) = att_1(u_1, @l)$. Therefore, $T_2 \not\models \beta.\tau.\tau_{@l}.@e \rightarrow \beta.\tau.\tau_{@l}$, which contradicts the fact that $T_2 \models \Sigma_U$. (2) If φ is a foreign key $\beta.\tau.@l \subseteq_{FK} \beta'.\tau'.@l'$, then $T_1 \not\models \beta'.\tau'.@l' \rightarrow \beta'.\tau'$ or there exists $u \in nodes(\beta.\tau)$ such that $att_1(u, @l) \notin values(\beta'.\tau'.@l')$ in T_1. In the former case, we reach a contradiction as in (1). In the latter case, assume that v is the child of u in T_2 of type $\tau_{@l}$ (u is a node of T_2 by definition of T_1). By definition of T_1, we have that $v \in nodes(\beta.\tau.\tau_{@l})$ and $att_2(v, @e) = att_1(u, @l)$. Thus, given that $values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2 is equal to $values(\beta'.\tau'.@l')$ in T_1, we conclude that $att_2(v, @e) \notin values(\beta'.\tau'.\tau'_{@l'}.@e)$ in T_2. Therefore, $T_2 \not\models \beta.\tau.\tau_{@l}.@e \subseteq_{FK} \beta'.\tau'.\tau'_{@l'}.@e$, which contradicts the fact that $T_2 \models \Sigma_U$. This concludes the proof of the lemma. $\square$

By Lemma B.2.1, from now on we consider only one-attribute DTDs. Let $D = (E, \{@l\}, P, R, r)$ be a one-attribute DTD and $D_N = (E_N, \{@l\}, P_N, R_N, r)$ be the narrow DTD of D (defined in the proof of Theorem 5.3.1). Observe that D_N is also one-attribute. Furthermore, observe that an XML tree T valid w.r.t. D may not conform to D_N and vice versa. Furthermore, an $\mathcal{AC}^{reg}_{K,FK}$-constraint φ over D may be satisfied by T but it may not be satisfied by any XML tree conforming to D_N. To explore the connection between XML trees conforming to D and those conforming to D_N, we replace $\mathcal{AC}^{reg}_{K,FK}$-constraints over D by new $\mathcal{AC}^{reg}_{K,FK}$-constraints over D_N. More precisely, given a set Σ of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, we define a set Σ_N of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D_N, referred to as the *narrowed set of constraints of* Σ, as follows. Let f be a substitution for the element types in E defined as $f(\tau) = \tau.(E_N - E)^*$ for every $\tau \in E$. Then for every key constraint $\beta.\tau.@l \to \beta.\tau$ in Σ, $f(\beta).\tau.@l \to f(\beta).\tau$ is in Σ_N, and for every foreign key constraint $\beta_1.\tau_1.@l \subseteq_{FK} \beta_2.\tau_2.@l$ in Σ (recall that $@l$ is the only attribute of D), $f(\beta_1).\tau_1.@l \subseteq_{FK} f(\beta_2).\tau_2.@l$ is in Σ_N.

We are now ready to establish the connection between D and D_N, which allows us to consider only narrow DTDs from now on.

Lemma B.2.2 *Let D be a one-attribute DTD, D_N the narrowed DTD of D, Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D and Σ_N the narrowed set of constraints of Σ. Then there exists an XML tree T_1 such that $T_1 \models (D, \Sigma)$ iff there exists an XML tree T_2 such that $T_2 \models (D_N, \Sigma_N)$.*

Proof: It suffices to show the following:

Claim: Given any XML tree $T_1 \models D$ one can construct an XML tree T_2 by modifying T_1 such that $T_2 \models D_N$, and vice versa. Furthermore, for any regular expression $\beta.\tau$ over D and $@l \in R(\tau)$, $|nodes(f(\beta).\tau)|$ in T_2 equals $|nodes(\beta.\tau)|$ in T_1, and $values(f(\beta).\tau.@l)$ in T_2 equals $values(\beta.\tau.@l)$ in T_1, where f is the substitution defined above.

For if the claim holds, we can show the lemma as follows. Assume that there exists an XML tree T_1 such that $T_1 \models (D, \Sigma)$. By the claim, there is T_2 such that $T_2 \models D_N$. Suppose, by contradiction, there is $\varphi \in \Sigma_N$ such that $T_2 \not\models \varphi$. (1) If φ is a key $f(\beta).\tau.@l \to f(\beta).\tau$, then there exist two distinct nodes $x, y \in nodes(f(\beta).\tau)$ in T_2 such that $x.@l = y.@l$. In other words, $|values(f(\beta).\tau.@l)| < |nodes(f(\beta).\tau)|$ in T_2. Since $T_1 \models \varphi$, it must be the case that $|values(\beta.\tau.@l)| = |nodes(\beta.\tau)|$ in T_1 because the value $x.@l$ of each $x \in nodes(\beta.\tau)$ uniquely identifies x among $nodes(\beta.\tau)$. This contradicts the claim that $|nodes(f(\beta).\tau)|$ in T_2 equals $|nodes(\beta.\tau)|$ in T_1 and $values(f(\beta).\tau.@l)$ in T_2 equals $values(\beta.\tau.@l)$ in T_1. (2) If φ is a foreign key: $f(\beta_1).\tau_1.@l \subseteq_{FK} f(\beta_2).\tau_2.@l$, then either $T_2 \not\models f(\beta_2).\tau_2.@l \to f(\beta_2).\tau_2$ or there is $x \in nodes(f(\beta_1).\tau_1)$ such that for all $y \in nodes(f(\beta_2).\tau_2)$ in T_2, $x.@l \neq y.@l$. In the first case, we reach a contradiction as in (1). In the second case, we have $x.@l \notin values(f(\beta_2).\tau_2.@l)$ in T_2. By the claim,

$x.@l \in values(\beta_1.\tau_1.@l)$ in T_1. Since $T_1 \models \varphi$, $x.@l \in values(\beta_2.\tau_2.@l)$ in T_1. Again by the claim, we have $x.@l \in values(f(\beta_2).\tau_2.@l)$ in T_2, which contradicts the assumption. The proof for the other direction is similar.

We next verify the claim. Given an XML tree $T_1 = (V_1, lab_1, ele_1, att, root)$ such that $T_1 \models D$, we construct an XML tree T_2 by modifying T_1 such that $T_2 \models D_N$. Consider a τ-element v in T_1. Let $ele_1(v) = [v_1, ..., v_n]$ and $w = lab_1(v_1) \ldots lab_1(v_n)$. Recall N_τ and P_τ, the set of nonterminals and the set of production rules generated when narrowing $\tau \rightarrow P(\tau)$ (see proof of Theorem 5.3.1). Let Q_τ be the set of E symbols that appear in P_τ. We can view $G = (Q_\tau, N_\tau \cup \{\tau\}, P_\tau, \tau)$ as an extended context free grammar, where Q_τ is the set of terminals, $N_\tau \cup \{\tau\}$ the set of nonterminals, P_τ the set of production rules and τ the start symbol[3]. Since $T_1 \models D$, we have $w \in P(\tau)$. By a straightforward induction on the structure of $P_N(\tau)$ it can be verified that w is in the language defined by G. Thus there is a parse tree $T(w)$ w.r.t. the grammar G for w, and w is the frontier (the list of leaves from left to right) of $T(w)$. Without loss of generality, assume that the root of $T(w)$ is v, and the leaves are $v_1, \ldots, v_n$. Observe that the internal nodes of $T(w)$ are labeled with element types in N_τ except that the root v is labeled τ. Intuitively, we construct T_2 by replacing each element v in T_1 by such a parse tree. More specifically, let $T_2 = (V_2, lab_2, ele_2, att, root)$. Here V_2 consists of nodes in V_1 and the internal nodes introduced in the parse trees. For each x in V_2, let $lab_2(x) = lab_1(x)$ if $x \in V_1$, and otherwise let $lab_2(x)$ be the node label of x in the parse tree where x belongs. Note that nodes in $V_2 - V_1$ are elements of some type in $E_N - E$. For every $x \in V_1$, let $ele_2(x)$ be the list of its children in the parse tree having x as root. For every $x \in V_2 - V_1$, let $ele_2(x)$ be the list of its children in the parse tree containing x. Note that att remains unchanged. By the construction of T_2 it can be verified that $T_2 \models D_N$; and moreover, for every regular expression $\beta.\tau$ over D and $@l \in R(\tau)$, $|nodes(f(\beta).\tau)|$ in T_2 equals $|nodes(\beta.\tau)|$ in T_1 and $values(f(\beta).\tau.@l)$ in T_2 equals $values(\beta.\tau.@l)$ in T_1 because, among other things, (1) if a string $r.\tau_1. \cdots .\tau_n.\tau$ over E is in $\beta.\tau$, then for every sequence of strings $w_0, \ldots, w_n$ in $(E_N - E)^*$, $r.w_0.\tau_1.w_1. \cdots .\tau_n.w_n.\tau$ is in $f(\beta).\tau$; (2) if a string $r.w_0.\tau_1.w_1. \cdots .\tau_n.w_n.\tau$ is in $f(\beta).\tau$, where $\tau_1, \ldots, \tau_n$, τ are element types in E and $w_0, \ldots, w_n$ are strings in $(E_N - E)^*$, then $r.\tau_1. \cdots .\tau_n.\tau$ is in $\beta.\tau$; (3) none of the new nodes, i.e., nodes in $V_2 - V_1$, is labeled with an E type; (4) no new attributes are defined; and (5) the ancestor-descendant relation on T_1-elements is not changed in T_2.

[3]As in the proof of Lemma B.1.1, if τ is in $P(\tau)$, then we need to rename τ in Q_τ to ensure that Q_τ and $N_\tau \cup \{\tau\}$ are disjoint. It is straightforward to handle that case.

Conversely, assume that there is $T_2 = (V_2,\, lab_2,\, ele_2,\, att,\, root)$ such that $T_2 \models D_N$. We construct an XML tree T_1 by modifying T_2 such that $T_1 \models D$. For any node $v \in V_2$ with $lab(v) = \tau$ and $\tau \in E_N - E$, we substitute v in $ele_2(v')$ by the children of v, where v' is the parent of v. In addition, we remove v from V_2, $lab_2(v)$ from lab_2, and $ele_2(v)$ from ele_2. Observe that by the definition of D_N, no attributes are defined for elements of any type in $E_N - E$. We repeat the process until there is no node labeled with element type in $E_N - E$. Now let $T_1 = (V_1,\, lab_1,\, ele_1,\, att,\, root)$, where V_1, lab_1 and ele_1 are V_2, lab_2 and ele_2 at the end of the process, respectively. Observe that att and $root$ remain unchanged. By the definition of T_1 it can be verified that $T_1 \models D$; and in addition, for any regular expression $\beta.\tau$ over D and $@l \in R(\tau)$, $|nodes(\beta.\tau)|$ in T_1 equals $|nodes(f(\beta).\tau)|$ in T_2, and $values(\beta.\tau.@l)$ in T_1 equals $values(f(\beta).\tau.@l)$ in T_2, because of (1) and (2) above and, among other things, the fact that none of the nodes removed is labeled with a type of E and the attribute function att is unchanged. □

We now move to encoding of DTDs, more specifically, narrow one-attribute DTDs. Let $D = (E,\, \{@l\},\, P,\, R,\, r)$ be a narrow one-attribute DTD and Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D. We encode D with a system Ψ^{Σ}_{D} of integer constraints such that there exists an XML tree conforming to D iff Ψ^{Σ}_{D} admits a nonnegative solution. The coding is developed w.r.t. Σ. More specifically, assume that $\beta_1.\tau_1.@l, \ldots, \beta_k.\tau_k.@l$ is an enumeration of all regular expressions and attributes that appear in Σ and Θ be the set of functions $\theta : \{1, \ldots, k\} \to \{0, 1\}$ which are not identically 0. For every $\theta \in \Theta$, define a regular expression:

$$r_\theta \;=\; \Big(\bigcap_{i\,:\,\theta(i)=1} \beta_i.\tau_i \Big) \cap \Big(\bigcap_{j\,:\,\theta(j)=0} \overline{\beta_j.\tau_j} \Big), \tag{B.1}$$

where $\overline{\beta_j.\tau_j}$ is the complement $\beta_j.\tau_j$. We allow intersection and complement operators only in regular expressions r_θ. We note that for every $i \in [1, k]$:[4]

$$\beta_i.\tau_i \;=\; \bigcup_{\theta\,:\,\theta(i)=1} r_\theta.$$

Then to capture the interaction between D and constraints of Σ, the system Ψ^{Σ}_{D} has a variable $|nodes(\beta_i.\tau_i)|$, for every $i \in [1, k]$, and $|nodes(r_\theta)|$, for every $\theta \in \Theta$. In other words, Ψ^{Σ}_{D} specifies the dependencies imposed by D on the number of elements reachable by following $\beta_i.\tau_i$ ($i \in [1, k]$) and r_θ ($\theta \in \Theta$).

[4]Recall that the regular language defined by a regular expression β is denoted by β as well.

To capture $\beta_i.\tau_i$ ($i \in [1,k]$) and r_θ ($\theta \in \Theta$) in Ψ_D^Σ, consider, for each regular expression $\beta_i.\tau_i$ ($i \in [1,k]$), a deterministic automaton that recognizes that expression. Let M be the deterministic automaton equivalent to the product of all these automata. We refer to M as the DFA for Σ. Let s_M be the start state of M and δ be its transition function. Given an XML tree T conforming to D, for each node x in T we define $state(x)$ as s, if there is a simple path ρ over D such that $T \models \rho(root, x)$ and $s = \delta(s_M, \rho)$. The connection between M and T w.r.t. $\beta_i.\tau_i$ ($i \in [1,k]$) is described by the following lemma:

Lemma B.2.3 *Let D be a narrow one-attribute DTD, Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, M the DFA for Σ and $\beta_i.\tau_i$ a regular expression in Σ. Then for every XML tree T conforming to D and every τ_i-element x in T, $x \in nodes(\beta_i.\tau_i)$ in T iff $state(x)$ contains some final state $f_{\beta_i.\tau_i}$ of the automaton for $\beta_i.\tau_i$.*

In other words, $nodes(\beta_i.\tau_i)$ in T consists of all τ_i-elements x such that $state(x)$ (which is a tuple of states of automata corresponding to regular expressions in Σ) contains some final state $f_{\beta_i.\tau_i}$ of the automaton for $\beta_i.\tau_i$. A similar idea was exploited in [AV99].

Proof: Since T is a tree, there exists a unique simple path ρ over D such that $T \models \rho.\tau_i(root,\ x)$. Thus $x \in nodes(\beta.\tau_i)$ in T iff $\rho.\tau_i \in \beta.\tau_i$. If $x \in nodes(\beta.\tau_i)$ in T, then $\rho.\tau_i \in \beta.\tau_i$ and, therefore, there must be a final state $f_{\beta.\tau_i}$ in the automaton for $\beta.\tau_i$ and a state s in M such that $s = \delta(s_M, \rho.\tau_i)$ and s contains $f_{\beta_i.\tau_i}$. Thus $state(x) = s$ contains some final state $f_{\beta_i.\tau_i}$ of the automaton for $\beta_i.\tau_i$. Conversely, if $state(x)$ contains a final state $f_{\beta_i.\tau_i}$ in the automaton for $\beta_i.\tau_i$, then $\rho.\tau_i \in \beta.\tau_i$ since $s = \delta(s_M, \rho.\tau_i)$. Therefore, $x \in nodes(\beta_i.\tau_i)$ in T. $\square$

We next define a system Ψ_D^Σ of integer constraints. The variables used in the constraints of Ψ_D^Σ are as follows. Let $\tau \in E$ be an element type and $s = \delta(s_M, \rho.\tau)$ for some simple path $\rho.\tau \in E^*$. For each such pair we create a distinct variable x_τ^s. Intuitively, in an XML tree T conforming to D, we use x_τ^s to keep track of the number of τ-elements with state s. Furthermore, define Y_τ^s as the set of pairs (τ', s') such that $\tau' \in E$, $s' = \delta(s_M, \rho.\tau')$ for some simple path $\rho.\tau' \in E^*$, τ is mentioned in $P(\tau')$ and $s = \delta(s', \tau)$. For each such pair (τ', s'), we create a variable $x^{s,s'}_{\tau,\tau'}$. Intuitively, in an XML tree T conforming to D, we use $x^{s,s'}_{\tau,\tau'}$ to keep track of the number of τ-elements with state s that are children of a node of type τ' with state s'. There are exponentially many variables (in the size of D and Σ) in total since M is a DFA. Using these, we define an integer constraint to specify $\tau \to P(\tau)$ at state s as follows. Let us use Ψ_τ^s to denote the set of integer constraints defined for τ at s.

- If $P(\tau) = \tau_1$, then Ψ_τ^s includes $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$, where $s_1 = \delta(s, \tau_1)$.
- If $P(\tau) = (\tau_1, \tau_2)$, then Ψ_τ^s includes $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$ and $x_\tau^s = x_{\tau_2,\tau}^{s_2,s}$, where $s_i = \delta(s, \tau_i)$ for $i = 1, 2$. Referring to the XML tree T, these assure that each τ-element in T must have a τ_1-subelement and a τ_2-subelement.
- If $P(\tau) = (\tau_1|\tau_2)$, then Ψ_τ^s includes $x_\tau^s = x_{\tau_1,\tau}^{s_1,s} + x_{\tau_2,\tau}^{s_2,s}$, where $s_i = \delta(s, \tau_i)$ for $i = 1, 2$. This assures that each τ-element in T must have either a τ_1-subelement or a τ_2-subelement, and thus the sum of the number of these τ_1-subelements and the number of τ_2-subelements equals the number of τ-elements.
- If $P(\tau) = \tau_1^*$, then Ψ_τ^s includes $(x_{\tau_1,\tau}^{s_1,s} > 0) \to (x_\tau^s > 0)$, where $s_1 = \delta(s, \tau_1)$.

In addition, Ψ_τ^s includes $x_\tau^s = \displaystyle\sum_{(\tau',s') \in Y_\tau^s} x_{\tau,\tau'}^{s,s'}$.

Recall that $\beta_1.\tau_1.@l, \ldots, \beta_k.\tau_k.@l$ is an enumeration of all regular expressions and attributes that appear in Σ, that Θ is the set of functions $\theta : \{1, \ldots, k\} \to \{0, 1\}$ which are not identically 0 and that for each such function θ, r_θ is a regular expression defined as in (B.1). For each $i \in [1, k]$, we define $F_{\beta_i.\tau_i}$ as the set of states $s = (s_1, \ldots, s_k)$ of the DFA for Σ such that s_i is a final state of the DFA for $\beta_i.\tau_i$. Notice that by Lemma B.2.3, for every XML tree T conforming to D and every node x of T, $x \in \mathit{nodes}(\beta_i.\tau_i)$ in T if and only if $\mathit{state}(x) \in F_{\beta_i.\tau_i}$. Furthermore, for each $\theta \in \Theta$, we define F_θ as the set of states $s = (s_1, \ldots, s_k)$ of the DFA for Σ such that for every $i \in [1, k]$, s_i is a final state of the DFA for $\beta_i.\tau_i$ if and only if $\theta(i) = 1$. Notice that by Lemma B.2.3, for every XML tree T conforming to D and every node x of T, $x \in \mathit{nodes}(r_\theta)$ in T if and only if $\mathit{state}(x) \in F_\theta$. Finally, for each $r_\theta \neq \emptyset$, we have that for every $i, j \in [1, k]$, if $\theta(i) = \theta(j) = 1$, then $\tau_i = \tau_j$. In this case, we define τ_θ as τ_i, for an arbitrary $i \in [1, k]$ such that $\theta(i) = 1$.

By our restriction on regular expressions regarding element type r, there is a unique variable x_r^s associated with r, where $s = \delta(s_M, r)$. We write x_r for x_r^s. Then we define *the set of cardinality constraints determined by DTD D* w.r.t. a set Σ of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, denoted by Ψ_D^Σ, as follows:

- For each $\tau \in E$ and each state s given above, Ψ_D^Σ contains all the constraints in Ψ_τ^s.

- Ψ_D^Σ contains constraint $x_r = 1$; i.e., there is a unique root in each XML tree conforming to D.

- For every $i \in [1, k]$, Ψ_D^Σ contains constraint $|nodes(\beta_i.\tau_i)| = \sum_{s \,:\, s \in F_{\beta_i.\tau_i}} x_{\tau_i}^s$.

- For every $\theta \in \Theta$ such that $r_\theta \neq \emptyset$, Ψ_D^Σ contains constraint $|nodes(r_\theta)| = \sum_{s \,:\, s \in F_\theta} x_{\tau_\theta}^s$.

- For every $\theta \in \Theta$ such that $r_\theta = \emptyset$, Ψ_D^Σ contains constraint $|nodes(r_\theta)| = 0$.

Note that Ψ_D^Σ can be computed in EXPTIME in the size of D and Σ. We say that Ψ_D^Σ is *consistent* iff it has a nonnegative solution. We next show that Ψ_D^Σ indeed characterizes narrow one-attribute DTD D.

Lemma B.2.4 *Let D be a narrow one-attribute DTD, Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D and Ψ_D^Σ the set of cardinality constraints determined by D w.r.t. Σ. Then Ψ_D^Σ is consistent iff there is an XML tree T such that $T \models D$. In addition, for every $i \in [1, k]$ and $\theta \in \Theta$, $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ in T equal the value of variables $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ given by the solution to Ψ_D^Σ.*

Proof: First, assume that there is an XML tree $T = (V, lab, ele, att, root)$ conforming to D. We define a nonnegative solution of Ψ_D^Σ as follows. For each variable $x_{\tau,\tau'}^{s,s'}$ in Ψ_D^Σ, let its value be the number of τ-elements x in T such that x is a child of a node y of type τ' with $state(x) = s$ and $state(y) = s'$. Furthermore, let x_r be 1 and for every variable x_τ^s in Ψ_D^Σ, let x_τ^s be the sum of the variables $x_{\tau,\tau'}^{s,s'}$ where $(\tau', s') \in Y_\tau^s$. Finally, for every $i \in [1, k]$ and every $\theta \in \Theta$, let $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ be $\sum_{s \,:\, s \in F_{\beta_i.\tau_i}} x_{\tau_i}^s$ and $\sum_{s \,:\, s \in F_\theta} x_{\tau_\theta}^s$, respectively. This defines a nonnegative assignment since T is finite. It can be verified that the assignment is a solution of Ψ_D^Σ. Indeed, it satisfies the constraint $x_r = 1$ and constraints of the form $x_\tau^s = \sum_{(\tau',s') \in Y_\tau^s} x_{\tau,\tau'}^{s,s'}$, $|nodes(\beta_i.\tau_i)| = \sum_{s \,:\, s \in F_{\beta_i.\tau_i}} x_{\tau_i}^s$ and $|nodes(r_\theta)| = \sum_{s \,:\, s \in F_\theta} x_{\tau_\theta}^s$ by the definition of the assignment. Moreover, one can verify that it also satisfies the constraints of each Ψ_τ^s, by considering four different cases corresponding to the four different types of regular expressions in D. In particular, it satisfies constraints of the form $(x_{\tau_1,\tau}^{s_1,s} > 0) \rightarrow (x_\tau^s > 0)$ for each $\tau \rightarrow \tau_1^*$ in P, since if $x_{\tau_1,\tau}^{s_1,s} > 0$, then there exists a τ_1-node in T having as its parent a τ-node y with $state(y) = s$. Thus, $x_\tau^s > 0$ by the definition of the assignment. Therefore, Ψ_D^Σ is consistent. Moreover, by Lemma B.2.3, for every $i \in [1, k]$ and $\theta \in \Theta$, the values of variables $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ in the solution are indeed $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ in T.

Conversely, assume that Ψ_D^Σ admits a nonnegative solution. We show that there exists an XML tree $T = (V, lab, ele, att, root)$ such that $T \models D$. To do so, for each element type τ and state s for τ, we create x_τ^s many distinct τ-elements. Let $ext(\tau)$ denote the set of all τ-elements created above and

$$V \;=\; \bigcup_{\tau \in E} ext(\tau).$$

Then function lab is defined as $lab(v) = \tau$ if $v \in ext(\tau)$, and function att is defined as follows:

$$att(v,\, @l) = \begin{cases} \text{empty string} & \text{if } @l \in R(lab(v)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

It is easy to verify that these functions are well defined. Let $root$ be the node labeled r, which is unique since $x_r = 1$ is in Ψ_D^Σ. Finally, to define function ele, we do the following. For each $x_{\tau,\tau'}^{s,s'}$ in Ψ_D^Σ, we choose $x_{\tau,\tau'}^{s,s'}$ many distinct vertices labeled τ and mark them with $x_{\tau,\tau'}^{s,s'}$. Note that every τ-element in V can be marked once and only once. Starting at $root$, for each τ-element x marked with $x_{\tau,\tau'}^{s,s'}$ for some $(\tau', s') \in Y_\tau^s$, consider $P(\tau)$ and constraints of Ψ_D^Σ [5]. If $P(\tau)$ is $\tau_1 \in E$, then we choose a distinct τ_1-element y marked with $x_{\tau_1,\tau}^{s_1,s}$ and let $ele(x) = [y]$, where $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$ is in Ψ_D^Σ. If $P(\tau) = (\tau_1, \tau_2)$, then we choose a τ_1-element y_1 marked with $x_{\tau_1,\tau}^{s_1,s}$ and a τ_2-element y_2 marked with $x_{\tau_2,\tau}^{s_2,s}$ and let $ele(x) = [y_1, y_2]$, where $x_\tau^s = x_{\tau_1,\tau}^{s_1,s}$ and $x_\tau^s = x_{\tau_2,\tau}^{s_2,s}$ are in Ψ_D^Σ. If $P(\tau) = (\tau_1|\tau_2)$, then we choose an element y marked with either $x_{\tau_1,\tau}^{s_1,s}$ or $x_{\tau_2,\tau}^{s_2,s}$ and let $ele(x) = [y]$, where $x_\tau^s = x_{\tau_1,\tau}^{s_1,s} + x_{\tau_2,\tau}^{s_2,s}$ is in Ψ_D^Σ. If $P(\tau) = \tau_1^*$, then we choose a list $[y_1, \ldots, y_n]$ $(n \geq 0)$ of τ_1-elements marked with $x_{\tau_1,\tau}^{s_1,s}$ and let $ele(x) = [y_1, \ldots, y_n]$, where $(x_{\tau_1,\tau}^{s_1,s} > 0) \rightarrow (x_\tau^s > 0)$ is in Ψ_D^Σ. By the constraints in Ψ_D^Σ, each element of V can be chosen once and only once. One can verify that T defined in this way is indeed an XML tree and $T \models D$. Hence, there exists an XML tree conforming to D.

Finally, to see that for every $i \in [1, k]$ and $\theta \in \Theta$, $|nodes(\beta_i.\tau_i)|$ and $|nodes(r_\theta)|$ in T equals the values of variables $|nodes(\beta.\tau)|$ and $|nodes(r_\theta)|$ in the solution, respectively, it suffices to show, by Lemma B.2.3, that for each node x in T, if x is marked with $x_{\tau,\tau'}^{s,s'}$ in the construction, then $state(x) = s$. Since T is a tree, there is a unique simple path

[5] We assume that $root$ is marked with x_r^s, where $s = \delta(s_M, r)$ and s_M is the initial state of the DFA for Σ.

$\rho \in E^*$ such that $T \models \rho(root, x)$. We show the claim by induction on the length $|\rho|$ of ρ. If $|\rho| = 1$, i.e., $\rho = r$, then x is the root and obviously, $state(x) = \delta(s_M, r)$. Assume the claim for ρ and we show that the claim holds for $\rho.\tau$. Let y be the τ'-element in T such that $T \models \rho(root, y)$ and y is the parent of x. Suppose that y is marked with $x^{s',s''}_{\tau',\tau''}$ in the construction. By the induction hypothesis $state(y) = s'$. It is easy to see $state(x) = \delta(s', \tau)$. By the definition of $\Psi^{s'}_{\tau'}$, we have that s is precisely the state $\delta(s', \tau)$. Thus $state(x) = s$. This proves the claim and thus the lemma. □

We now move to encoding $\mathcal{AC}^{reg}_{K,FK}$-constraints in terms of integer constraints. Let D be a DTD (E, $\{@l\}$, P, R, r) and Σ a set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D. By Lemmas B.2.1 and B.2.2, we assume, without loss of generality, that D is a narrow one-attribute DTD. To encode Σ, let $\beta_1.\tau_1.@l, \ldots, \beta_k.\tau_k.@l$ be an enumeration of all regular expressions and attributes that appear in Σ, and for every function $\theta : \{1, \ldots, k\} \to \{0, 1\}$ which is not identically 0, let regular expression r_θ be defined as in (B.1). Then for every nonempty $\Omega \subseteq \Theta$, we introduce a new variables z_Ω. In any XML tree conforming to D, the intended interpretations of z_Ω is the the cardinality of

$$\Big(\bigcap_{\theta\,:\,\theta\in\Omega} values(r_\theta.@l)\Big) - \Big(\bigcup_{\theta\,:\,\theta\in\Theta-\Omega} values(r_\theta.@l)\Big). \tag{B.2}$$

Note that the number of new variables is double-exponential in the number of regular expression in Σ. Using these variables, we define the set of *the cardinality constraints determined by* Σ, denoted by C_Σ, which consists of the following:

$$\sum_{\Omega\,:\,\theta\in\Omega} z_\Omega = |values(r_\theta.@l)| \qquad \text{for every } \theta \in \Theta,$$

$$\sum_{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset} z_\Omega = |values(\beta_i.\tau_i.@l)| \qquad \text{for every } i \in [1,k],$$

$$|values(\beta_i.\tau_i.@l)| = |nodes(\beta_i.\tau_i)| \qquad \text{for every } \beta_i.\tau_i.@l \to \beta_i.\tau_i \text{ in } \Sigma,$$

$$|values(\beta_j.\tau_j.@l)| = |nodes(\beta_j.\tau_j)| \qquad \text{for every } \beta_i.\tau_i.@l \subseteq_{FK} \beta_j.\tau_j.@l \text{ in } \Sigma,$$

$$\sum_{\substack{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset,\\ \Omega\cap\{\theta'|\theta'(j)=1\}=\emptyset}} z_\Omega = 0 \qquad \text{for every } \beta_i.\tau_i.@l \subseteq_{FK} \beta_j.\tau_j.@l \text{ in } \Sigma,$$

$$|values(\beta_i.\tau_i.@l)| \leq |nodes(\beta_i.\tau_i)| \qquad \text{for every } i \in [1,k],$$

$$|values(r_\theta.@l)| \leq |nodes(r_\theta)| \qquad \text{for every } \theta \in \Theta.$$

Note that the size of C_Σ is double-exponential in the size of Σ.

We now combine the encodings for constraints and the DTDs, and present a system $\Psi(D, \Sigma)$ of linear integer constraints for a DTD D and a set Σ of $\mathcal{AC}^{reg}_{K,FK}$-constraints. Assuming that D and Σ are as above, the set $\Psi(D, \Sigma)$, called the *set of cardinality constraints determined by* D *and* Σ, is defined to be:

$$\Psi_D^\Sigma \ \cup \ C_\Sigma \ \cup \ \{(|nodes(\beta_i.\tau_i)| > 0) \to (|values(\beta_i.\tau_i.@l)| > 0) \mid i \in [1,k]\} \ \cup$$
$$\{(|nodes(r_\theta)| > 0) \to (|values(r_\theta.@l)| > 0) \mid \theta \in \Theta\},$$

where C_Σ is the set of cardinality constraints determined by Σ, and Ψ_D^Σ is the set of cardinality constraints determined by D w.r.t. Σ. The system $\Psi(D, \Sigma)$ is said to be *consistent* iff it has a nonnegative solution that satisfies all of its constraints. Observe that $\Psi(D, \Sigma)$ can be partitioned into two sets: $\Psi(D, \Sigma) = \Psi^l(D, \Sigma) \cup \Psi^d(D, \Sigma)$, where $\Psi^l(D, \Sigma)$ consists of linear integer constraints, and $\Psi^d(D, \Sigma)$ consists of constraints of the form $(x > 0 \to y > 0)$. Also note that the size of $\Psi(D, \Sigma)$ is double-exponential in the size of D and Σ.

We next show that $\Psi(D, \Sigma)$ indeed characterizes the consistency of D and Σ.

Lemma B.2.5 *Let D be a narrow one-attribute DTD, Σ a finite set of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D and $\Psi(D, \Sigma)$ the set of cardinality constraints determined by D and Σ. Then $\Psi(D, \Sigma)$ is consistent if and only if there is an XML tree T such that $T \models (D, \Sigma)$.*

Proof: Suppose that there exists an XML tree T such that $T \models (D, \Sigma)$. Then by Lemma B.2.4, there exists a nonnegative solution for Ψ_D^Σ such that for every $i \in [1, k]$ and $\theta \in \Theta$, the values of variables $|nodes(r_\theta)|$ and $|nodes(\beta_i.\tau_i)|$ in this solution coincide with $|nodes(r_\theta)|$ and $|nodes(\beta_i.\tau_i)|$ in T, respectively. From this solution, it is easy to generate a solution to $\Psi(D, \Sigma)$ by assigning to variable $|values(r_\theta.@l)|$ the size of $values(r_\theta.@l)$ in T, for every $\theta \in \Theta$, assigning to variable $|values(\beta_i.\tau_i.@l)|$ the size of $values(\beta_i.\tau_i.@l)$ in T, for every $i \in [1, k]$, and then assigning to each variable z_Ω the cardinality of set (B.2) above. It is straightforward to verify that this assignment is a solution to $\Psi(D, \Sigma)$.

Conversely, suppose that $\Psi(D, \Sigma)$ has an integer solution. We show that there is an XML tree T such that $T \models (D, \Sigma)$. By Lemma B.2.4, given an integer solution to $\Psi(D, \Sigma)$, we can construct an XML tree $T' = (V, lab, ele, att', root)$ such that $T' \models D$. Moreover, for every $i \in [1, k]$, there are exactly $n_{\beta_i.\tau_i}$ elements in T' reachable by following $\beta_i.\tau_i$, where $n_{\beta_i.\tau_i}$ is the value of the variable $|nodes(\beta_i.\tau_i)|$ in $\Psi(D, \Sigma)$, and for every $\theta \in \Theta$, there are exactly n_{r_θ} elements in T' reachable by following r_θ, where n_{r_θ} is the value of the variable $|nodes(r_\theta)|$ in $\Psi(D, \Sigma)$. We modify the definition of the function att', while leaving V, lab, ele and $root$ unchanged, to generate a tree $T = (V, lab, ele, att, root)$ such that $T \models (D, \Sigma)$. More specifically, we modify $att'(v, @l)$ if v is in $nodes(\beta.\tau)$ for some regular expression $\beta.\tau$ mentioned in Σ, and leave $att'(v, @l)$ unchanged otherwise. To do this, for each variable z_Ω we create a set s_Ω of distinct string values such that $|s_\Omega| = z_\Omega$ and $s_\Omega \cap s_{\Omega'} = \emptyset$ if $\Omega \neq \Omega'$. Then for every $\Omega \subseteq \Theta$, we let $values(r_\theta.@l)$ in T to contain s_Ω if and only if $\theta \in \Omega$. This is possible because (1) $\sum_{\Omega\,:\,\theta\in\Omega} z_\Omega = |values(r_\theta.@l)|$ is in C_Σ, for every $\theta \in \Theta$; (2) $\sum_{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset} z_\Omega = |values(\beta_i.\tau_i.@l)|$ is in C_Σ, for every $i \in [1, k]$; (3) if $r_\theta = \emptyset$, then $|nodes(r_\theta)| = 0$ is in Ψ_D^Σ, for every $\theta \in \Theta$; (4) $|values(\beta_i.\tau_i.@l)| \leq |nodes(\beta_i.\tau_i)|$ is in C_Σ, for every $i \in [1, k]$; (5) $|values(r_\theta.@l)| \leq |nodes(r_\theta)|$ is in C_Σ, for every $\theta \in \Theta$; and (6) $nodes(\beta)$ in T equals $nodes(\beta)$ in T', for every regular expression β over D.

We next show that T has the desired properties. It is easy to verify $T \models D$ given the construction of T from T' and the assumption $T' \models D$. By definition of T, we have that for every $i \in [1, k]$ and $\theta \in \Theta$, $|nodes(\beta_i.\tau_i)|$, $|values(\beta_i.\tau_i.@l)|$, $|nodes(r_\theta)|$ and $|values(r_\theta.@l)|$ in T equal the value of variables $|nodes(\beta_i.\tau_i)|$, $|values(\beta_i.\tau_i.@l)|$,

$|nodes(r_\theta)|$ and $|values(r_\theta.@l)|$ given by the solution to $\Psi(D, \Sigma)$. We use this property to show that $T \models \Sigma$. Let φ be a constraint in Σ. (1) If φ is a key $\beta_i.\tau_i.@l \to \beta_i.@l$, it is immediate from the definition of T that $T \models \varphi$ since $|values(\beta_i.\tau_i.@l)| = |nodes(\beta_i.\tau_i)|$ is a constraint in C_Σ and, hence, $|values(\beta_i.\tau_i.@l)| = |nodes(\beta_i.\tau_i)|$ in T. That is, each $x \in nodes(\beta_i.\tau_i)$ in T has a distinct $@l$-attribute value and thus the value of its $@l$-attribute uniquely identifies x among nodes in $nodes(\beta_i.\tau_i)$. (2) If φ is $\beta_i.\tau_i.@l \subseteq_{FK} \beta_j.\tau_j.@l$, it is easy to see that in T:

$$values(\beta_i.\tau_i.@l) - values(\beta_j.\tau_j.@l) \;=\; \bigcup_{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset,\,\Omega\cap\{\theta'|\theta'(j)=1\}=\emptyset} s_\Omega,$$

Since $s_\Omega \cap s_{\Omega'} = \emptyset$ if $\Omega \neq \Omega'$,

$$|values(\beta_i.\tau_i.@l) - values(\beta_j.\tau_j.@l)| \;=\; \sum_{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset,\,\Omega\cap\{\theta'|\theta'(j)=1\}=\emptyset} z_\Omega.$$

Thus, given that

$$\sum_{\Omega\,:\,\Omega\cap\{\theta|\theta(i)=1\}\neq\emptyset,\,\Omega\cap\{\theta'|\theta'(j)=1\}=\emptyset} z_\Omega \;=\; 0$$

is in C_Σ (since $\beta_i.\tau_i.@l \subseteq_{FK} \beta_j.\tau_j.@l \in \Sigma$), we have $|values(\beta_i.\tau_i.@l) - values(\beta_j.\tau_j.@l)| = 0$ in T, that is, $values(\beta_i.\tau_i.@l) \subseteq values(\beta_j.\tau_j.@l)$ in T. Furthermore, $T \models \beta_j.\tau_j.@l \to \beta_j.\tau_j$ since $|values(\beta_j.\tau_j.@l)| = |nodes(\beta_j.\tau_j)|$ is a constraint in C_Σ. Thus $T \models \varphi$. This concludes the proof of the lemma. $\square$

We need another lemma for a mild generalization of linear integer constraints.

Lemma B.2.6 *Given a system $A\vec{x} \leq \vec{b}$ of linear integer constraints together with conditions of the form $(x_i > 0) \to (x_j > 0)$, where A is an $n \times m$ matrix on integers, $\vec{b}$ is an n-vector on integers and $1 \leq i, j \leq m$, the problem of determining whether the system admits a nonnegative integer solution is in NP.*

Proof: Let $c_1, \ldots, c_p$ enumerate the conditions of the form $(x > 0) \to (y > 0)$, c_k being $(x_k^1 > 0) \to (x_k^2 > 0)$. Consider 2^p instances $\mathcal{I}_j$ of integer linear programming obtained by adding, for each $k \leq p$, either $x_k^1 = 0$, or $x_k^2 > 0$ to $A\vec{x} \leq \vec{b}$. Clearly, the original system of constraints has a solution iff some $\mathcal{I}_j$ has a solution. By [Pap81], $\mathcal{I}_j$ has a solution iff it has a solution whose size is polynomial in A, $\vec{b}$ and p. Hence, to check if the original system of constraints has a solution, it suffices to guess a system $\mathcal{I}_j$ and then guess a polynomial size solution for it; thus, the problem is in NP. $\square$

We now conclude the proof of the first part of the theorem. By Lemma B.2.1, given an arbitrary DTD D and a set Σ of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D, it is possible to compute a one-attribute DTD D' and a set Σ' of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D' such that (D, Σ) is consistent iff (D', Σ') is consistent. By Lemma B.2.2, one can compute a narrow one-attribute DTD D'_N and a set Σ'_N of $\mathcal{AC}^{reg}_{K,FK}$-constraints over D'_N such that (D', Σ') is consistent iff (D'_N, Σ'_N) is consistent. By Lemma B.2.5, (D'_N, Σ'_N) is consistent iff $\Psi(D'_N, \Sigma'_N)$ has a nonnegative integer solution. Thus, (D, Σ) is consistent iff $\Psi(D'_N, \Sigma'_N)$ has a nonnegative integer solution. Note that (D', Σ') can be computed in polynomial time on $|D| + |\Sigma|$, (D'_N, Σ'_N) can be computed in polynomial time on $|D'| + |\Sigma'|$, and $\Psi(D'_N, \Sigma'_N)$ can be computed in double-exponential time on $|D'_N| + |\Sigma'_N|$. Thus, by Lemma B.2.6, one can check in 2-NEXPTIME whether there exists an XML tree T such that $T \models (D, \Sigma)$.

Proof of b) We establish the PSPACE-hardness by reduction from the QBF-CNF problem. An instance of QBF-CNF is a quantified boolean formula in prenex conjunctive normal form. The problem is to determine whether this formula is valid. QBF-CNF is known to be PSPACE-complete [GJ79, Pap94].

Let θ be a formula of the form

$$Q_1 x_1 \cdots Q_m x_m \psi, \tag{B.3}$$

where each $Q_i \in \{\forall, \exists\}$ $(1 \leq i \leq m)$ and ψ is a propositional formula in conjunctive normal form, say $C_1 \wedge \cdots \wedge C_n$, that mentions variables $x_1, \ldots, x_m$. We construct a DTD D_θ and a set Σ_θ of $\mathcal{AC}^{reg}_{K,FK}$-constraint such that θ is valid if and only if there is an XML tree conforming to D_θ and satisfying Σ_θ.

We construct a DTD $D_\theta = (E, A, P, R, r)$ as follows. $E = \{r, C\} \cup \bigcup_{i=1}^{m} \{x_i, \bar{x}_i, N_{x_i}, P_{x_i}\}$, $A = \{@l\}$ and P is defined by considering the quantifiers of θ. We use Q_1 to define P on the root:

$$P(r) = \begin{cases} (N_{x_1} | P_{x_1}), C & Q_1 = \exists \\ (N_{x_1}, P_{x_1}), C & Q_1 = \forall \end{cases}$$

In general, for each $1 \leq i \leq m - 1$, we consider quantifier Q_{i+1} to define $P(N_{x_i})$ and $P(P_{x_i})$:

$$P(N_{x_i}) = P(P_{x_i}) = \begin{cases} N_{x_{i+1}} | P_{x_{i+1}} & Q_{i+1} = \exists \\ N_{x_{i+1}}, P_{x_{i+1}} & Q_{i+1} = \forall \end{cases}$$

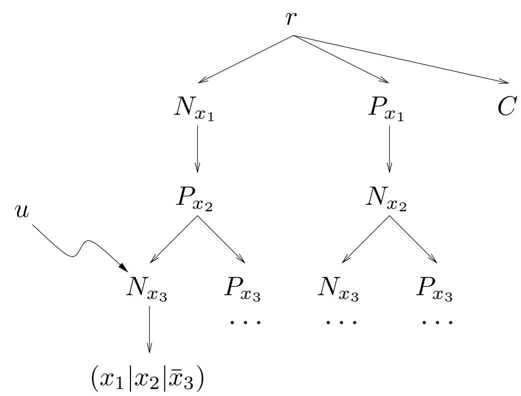


Figure B.2: An XML tree conforming to the DTD constructed from $\forall x_1 \exists x_2 \forall x_3 (x_1 \vee x_2 \vee \neg x_3)$.

We represent formula ψ as a regular expression. Given a clause $C_j = \bigvee_{i=1}^{p} y_i \vee \bigvee_{i=1}^{q} \neg z_i$ ($j \in [1, n]$), $tr(C_j)$ is defined to be the regular expression $y_1| \cdots |y_p|\bar{z}_1| \cdots |\bar{z}_q$. We define P on element types N_{x_m} and P_{x_m} as $P(N_{x_m}) = P(P_{x_m}) = tr(C_1), \ldots, tr(C_n)$. For the remaining elements of E, we define P as ϵ. We define function R as follows:

$$\begin{array}{lclclcll} R(r) & = & R(P_{x_i}) & = & R(N_{x_i}) & = & \emptyset & 1 \leq i \leq m \\ R(C) & = & R(x_i) & = & R(\bar{x}_i) & = & \{@l\} & 1 \leq i \leq m. \end{array}$$

Finally, Σ_θ contains the following foreign keys:

$$r.\_^*.N_{x_i}.\_^*.x_i.@l \subseteq_{FK} r.C.C.@l, \qquad r.\_^*.P_{x_i}.\_^*.\bar{x}_i.@l \subseteq_{FK} r.C.C.@l, \qquad i \in [1, m].$$

For instance, for the formula $\forall x_1 \exists x_2 \forall x_3 (x_1 \vee x_2 \vee \neg x_3)$, an XML tree conforming to D is shown in Figure B.2. In this tree, a node of type N_{x_i} represents a negative value (0) for the variable x_i and a node of type P_{x_i} represents a positive value (1) for this variable. Thus, given that the root has two children of types N_{x_1} and P_{x_1}, the values 0 and 1 are assigned to x_1 (representing the quantifier $\forall x_1$). Nodes of type N_{x_1} have one child of type either N_{x_2} or P_{x_2}, and, therefore, either 0 or 1 is assigned to x_2 (representing the quantifier $\exists x_2$). The same holds for nodes of type P_{x_2}. The fourth level of the tree represents the quantifier $\forall x_3$.

In Figure B.2, every path from the root r to a node of type either N_{x_3} or P_{x_3} represents a truth assignment for the variables x_1, x_2, x_3. For example, the path from the root to the node u represents the truth assignment σ_u: $\sigma_u(x_1) = 0$, $\sigma_u(x_2) = 1$ and $\sigma_u(x_3) = 0$. To verify that all these assignments satisfy the formula $x_1 \vee x_2 \vee \neg x_3$ we use the set of constraint Σ_θ.

Next we prove that θ, defined in (B.3), is valid if and only if there is an XML tree T conforming to D_θ and satisfying Σ_θ. We show only the "if" direction. The "only if" direction is similar.

Suppose that there is an XML tree T such that $T \models (D_\theta, \Sigma_\theta)$. To prove that θ is valid, it suffices to prove that each path from the root r to a node of type either N_{x_m} or P_{x_m} represents a truth assignment satisfying ψ. Let p be one of these paths and let v be the node of type either N_{x_m} or P_{x_m} reachable from the root by following p. We define the truth assignment σ_p as follows:

$$\sigma_p(x_i) = \begin{cases} 1 & p \text{ contains a node of type } P_{x_i} \\ 0 & \text{Otherwise.} \end{cases}$$

We have to prove that $\sigma_p(C_i) = 1$ for each $i \in [1, n]$. Given that $T \models D_\theta$, v has as a child a node v' whose type is in $tr(C_i)$. If the type of v' is x_j, then given that $T \models r.\_^*.N_{x_j}.\_^*.x_j.@l \subseteq_{FK} r.C.C.@l$ and that there is no a node in T reachable by following the path $r.C.C$, p contains a node of type P_{x_j}, and, therefore, $\sigma_p(C_i) = 1$ since $\sigma_p(x_j) = 1$. If the type of v' is $\bar{x}_j$, then given that $T \models r.\_^*.P_{x_j}.\_^*.\bar{x}_j.@l \subseteq_{FK} r.C.C.@l$, p contains a node of type N_{x_j} and it does not contain a node of type P_{x_j}, and, therefore, $\sigma_p(C_i) = 1$ since $\sigma_p(\neg x_j) = 1$. Thus, we conclude that θ is valid. This concludes the proof of part b) of the theorem.

B.3 Proof of Theorem 5.4.1

We establish the undecidability of the consistency problem for unary relative keys and foreign keys by reduction from the Hilbert's 10th problem [Mat93]. To do this, we consider a variation of the Diophantine problem, referred as the *positive Diophantine quadratic system problem*. An instance of the problem is

$$\begin{aligned} P_1(x_1, \ldots, x_k) &= Q_1(x_1, \ldots, x_k) + c_1 \\ P_2(x_1, \ldots, x_k) &= Q_2(x_1, \ldots, x_k) + c_2 \\ &\cdots \\ P_n(x_1, \ldots, x_k) &= Q_n(x_1, \ldots, x_k) + c_n \end{aligned}$$

where for $1 \leq i \leq n$, P_i and Q_i are polynomials in which all coefficients are positive integers; the degree of P_i is at most 2 and the degree of each of its monomial is at least

1; each polynomial Q_i satisfies the same condition, and each c_i is a nonnegative integer constant. The problem is to determine, given any positive Diophantine quadratic system, whether it has a nonnegative integer solution.

The positive Diophantine quadratic system problem is undecidable. To prove this, it is straightforward to reduce to it another variation of the Diophantine problem, the *positive Diophantine equation problem*, which is known to be undecidable. An instance of this problem is $R(\bar{y}) = S(\bar{y})$, where R and S are polynomials in which all coefficients are positive integers, and the problem is to determine whether it has a nonnegative integer solution.

In what follows, we show a reduction from the positive Diophantine quadratic system problem to $\mathsf{SAT}(\mathcal{RC}_{K,FK})$. More precisely, given a quadratic equation we show how to represent it by using a DTD and a set of constraints. It is straightforward to extend this representation to consider an arbitrary number of quadratic equations.

Consider the following equation:

$$\sum_{i=1}^{m} a_i x_{\alpha_i} + \sum_{i=m+1}^{n} a_i x_{\alpha_i} x_{\beta_i} \;=\; \sum_{i=1}^{p} b_i x_{\gamma_i} + \sum_{i=p+1}^{q} b_i x_{\gamma_i} x_{\delta_i} + o. \tag{B.4}$$

In this equation, for every $i \in [1, n]$ and $j \in [m + 1, n]$, a_i is a positive integer and x_{α_i}, x_{β_j} represent variables, where $\alpha_i, \beta_j \in [1, k]$. Furthermore, for every $i \in [1, q]$ and $j \in [p+1, q]$, b_i is a positive integer and x_{γ_i}, x_{δ_j} are variables, where $\gamma_i, \delta_j \in [1, k]$. Finally, o is a nonnegative integer.

To code the previous equation, we need to define a DTD $D = (E, A, P, R, r)$ and a set of $\mathcal{RC}_{K,FK}$-constraints Σ. Here D includes the following elements types:

$$\begin{aligned} E \;=\; & \{r, X, Y\} \cup \bigcup_{i=1}^{k} \{n_i\} \cup \\ & \bigcup_{i=1}^{n} \{\alpha_i\} \cup \bigcup_{i=m+1}^{n} \{\alpha'_i, \beta_i, c_i, d_i, e_i\} \cup \bigcup_{i=1}^{q} \{\gamma_i\} \cup \bigcup_{i=p+1}^{q} \{\gamma'_i, \delta_i, f_i, g_i, h_i\}, \end{aligned}$$

and it includes the following attributes: $A = \{@v\}$. In this DTD, r is the root. Intuitively, referring to an XML tree conforming to D, we use $|ext(n_i)|$ to code the value of the variable x_i, and we use $|ext(X)|$ and $|ext(Y)|$ to code the values of the left and the right hand sides of (B.4), respectively.

We define $P(r)$ as follows:

$$P(r) = n_1^*, \ldots, n_k^*, \alpha_1^*, \ldots, \alpha_m^*, (\epsilon|\alpha_{m+1}), \ldots, (\epsilon|\alpha_n), \gamma_1^*, \ldots, \gamma_p^*, (\epsilon|\gamma_{p+1}), \ldots, (\epsilon|\gamma_q), \underbrace{Y, \ldots, Y}_{o \text{ times}}$$

We define the function P on α_i and β_i as follows:

$$\begin{array}{rcll}
P(\alpha_i) & = & \underbrace{X, \ldots, X}_{a_i \text{ times}} & 1 \le i \le m \\
P(\alpha_i) & = & (\beta_i, c_i, c_i, \underbrace{X, \ldots, X}_{a_i \text{ times}})^*, \alpha'_i & m+1 \le i \le n \\
P(\gamma_i) & = & \underbrace{Y, \ldots, Y}_{b_i \text{ times}} & 1 \le i \le p \\
P(\gamma_i) & = & (\delta_i, f_i, f_i, \underbrace{Y, \ldots, Y}_{b_i \text{ times}})^*, \gamma'_i & p+1 \le i \le q
\end{array}$$

To code (B.4) we need to capture the multiplication operator. To do this, we use α'_i and γ'_i:

$$\begin{array}{rcll}
P(\alpha'_i) & = & (\beta_i, d_i, d_i)^*, (\alpha_i|(c_i, e_i)^*) & m+1 \le i \le n \\
P(\gamma'_i) & = & (\delta_i, g_i, g_i)^*, (\gamma_i|(f_i, h_i)^*) & p+1 \le i \le q
\end{array}$$

For all the other element types τ in D, $P(\tau)$ is defined as ϵ:

$$\begin{array}{rcll@{\qquad}rcll}
P(\beta_i) & = & \epsilon & m+1 \le i \le n & P(\delta_i) & = & \epsilon & p+1 \le i \le q \\
P(c_i) & = & \epsilon & m+1 \le i \le n & P(f_i) & = & \epsilon & p+1 \le i \le q \\
P(d_i) & = & \epsilon & m+1 \le i \le n & P(g_i) & = & \epsilon & p+1 \le i \le q \\
P(e_i) & = & \epsilon & m+1 \le i \le n & P(h_i) & = & \epsilon & p+1 \le i \le q \\
P(X) & = & \epsilon & & P(Y) & = & \epsilon & \\
P(n_i) & = & \epsilon & 1 \le i \le k & & & &
\end{array}$$

Finally, we include the following attributes:

$$\begin{array}{ll@{\qquad}ll}
R(r) = \emptyset & & R(\beta_i) = R(c_i) = R(d_i) = R(e_i) = \{@v\} & m+1 \le i \le n \\
R(n_i) = \{@v\} & 1 \le i \le k & R(\delta_i) = R(f_i) = R(g_i) = R(h_i) = \{@v\} & p+1 \le i \le q \\
R(\alpha_i) = \{@v\} & 1 \le i \le n & R(\alpha'_i) = \emptyset & m+1 \le i \le n \\
R(\gamma_i) = \{@v\} & 1 \le i \le q & R(\gamma'_i) = \emptyset & p+1 \le i \le q \\
R(X) = \{@v\} & & R(Y) = \{@v\} &
\end{array}$$

To ensure that XML documents that conform to D indeed code equation (B.4) we need to define a set of $\mathcal{RC}_{K,FK}$-constraints Σ. This set contains the following absolute keys:

$r(X.@v \to X)$		$r(Y.@v \to Y)$	
$r(\alpha_i.@v \to \alpha_i)$	for every $1 \leq i \leq n$	$r(\gamma_i.@v \to \gamma_i)$	for every $1 \leq i \leq q$
$r(\beta_i.@v \to \beta_i)$	for every $m+1 \leq i \leq n$	$r(\delta_i.@v \to \delta_i)$	for every $p+1 \leq i \leq q$
$r(c_i.@v \to c_i)$	for every $m+1 \leq i \leq n$	$r(f_i.@v \to f_i)$	for every $p+1 \leq i \leq q$
$r(d_i.@v \to d_i)$	for every $m+1 \leq i \leq n$	$r(g_i.@v \to g_i)$	for every $p+1 \leq i \leq q$
$r(e_i.@v \to e_i)$	for every $m+1 \leq i \leq n$	$r(h_i.@v \to h_i)$	for every $p+1 \leq i \leq q$
$r(n_i.@v \to n_i)$	for every $1 \leq i \leq k$		

Σ contains the following absolute foreign keys:

$r(X.@v \subseteq_{FK} Y.@v)$,	$r(Y.@v \subseteq_{FK} X.@v)$	
$r(n_s.@v \subseteq_{FK} \alpha_i.@v)$,	$r(\alpha_i.@v \subseteq_{FK} n_s.@v)$	$1 \leq i \leq n$ and the value of α_i in (B.4) is equal to s
$r(n_s.@v \subseteq_{FK} e_i.@v)$,	$r(e_i.@v \subseteq_{FK} n_s.@v)$	$m+1 \leq i \leq n$ and the value of β_i in (B.4) is equal to s
$r(n_s.@v \subseteq_{FK} \gamma_i.@v)$,	$r(\gamma_i.@v \subseteq_{FK} n_s.@v)$	$1 \leq i \leq q$ and the value of γ_i in (B.4) is equal to s
$r(n_s.@v \subseteq_{FK} h_i.@v)$,	$r(h_i.@v \subseteq_{FK} n_s.@v)$	$p+1 \leq i \leq q$ and the value of δ_i in (B.4) is equal to s

Finally, Σ contains the following relative foreign keys:

$\alpha_i(\beta_i.@v \subseteq_{FK} d_i.@v)$,	$\alpha_i(d_i.@v \subseteq_{FK} \beta_i.@v)$	$m+1 \leq i \leq n$
$\alpha'_i(\beta_i.@v \subseteq_{FK} c_i.@v)$,	$\alpha'_i(c_i.@v \subseteq_{FK} \beta_i.@v)$	$m+1 \leq i \leq n$
$\gamma_i(\delta_i.@v \subseteq_{FK} g_i.@v)$,	$\gamma_i(g_i.@v \subseteq_{FK} \delta_i.@v)$	$p+1 \leq i \leq q$
$\gamma'_i(\delta_i.@v \subseteq_{FK} f_i.@v)$,	$\gamma'_i(f_i.@v \subseteq_{FK} \delta_i.@v)$	$p+1 \leq i \leq q$

We show next that there is an XML tree T such that $T \models (D, \Sigma)$ if and only if there exists a nonnegative integer solution for (B.4). To do this, we prove that every XML tree T satisfying D and Σ codifies equation (B.4). More precisely, if the value of every variable x_i is v_i and $|ext(n_i)| = v_i$, for $i \in [1, k]$, then

$$|ext(X)| = \sum_{i=1}^{m} a_i v_{\alpha_i} + \sum_{i=m+1}^{n} a_i v_{\alpha_i} v_{\beta_i}, \tag{B.5}$$

$$|ext(Y)| = \sum_{i=1}^{p} b_i v_{\gamma_i} + \sum_{i=p+1}^{q} b_i v_{\gamma_i} v_{\delta_i} + o. \tag{B.6}$$

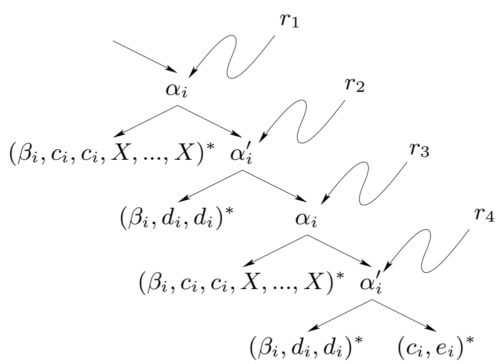


Figure B.3: Part of the XML tree used in the proof of Theorem 5.4.1.

Let T be an XML tree conforming to D. Then every node of type X in T appears as a child of some node of type α_i ($i \in [1, n]$). Thus, to prove (B.5) it suffices to show that the number of X-nodes that are children of some node of type α_i ($i \in [1, n]$) is equal to the i-th term of (B.5), that is, for every $i \in [1, m]$:

$$|\{x \mid x \text{ is an } X\text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i\}| \;=\; a_i v_{\alpha_i},$$

and for every $i \in [m + 1, n]$:

$$|\{x \mid x \text{ is an } X\text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i\}| \;=\; a_i v_{\alpha_i} v_{\beta_i}.$$

Analogously, to show that (B.6) holds, we have to prove that the number of Y-nodes that are children of some node of type γ_i ($i \in [1, q]$) is equal to the i-th term of (B.6). We will only consider here the case of X-nodes, being the other case similar.

First, let $i \in [1, m]$ and s be the value of α_i in (B.5). Given that $r(n_s.@v \subseteq_{FK} \alpha_i.@v)$, $r(\alpha_i.@v \subseteq_{FK} n_s.@v)$ are in Σ, by definition of $P(\alpha_i)$ the total number of X-nodes that are children of a node of type α_i is equal to $a_i v_{\alpha_i}$. Second, let $i \in [m + 1, n]$ and s, t be the values of α_i and β_i in (B.4), respectively. Next we prove that $|\{x \mid x \text{ is an } X\text{-node in } T \text{ and } x \text{ is a child of a node of type } \alpha_i\}| = a_i v_s v_t$.

Given that $r(n_s.@v \subseteq_{FK} \alpha_i.@v)$, $r(\alpha_i.@v \subseteq_{FK} n_s.@v)$ are in Σ, $|ext(\alpha_i)|$ in T is equal to $|ext(n_s)| = v_s$. Thus, in T there are exactly v_s nodes of type α_i, each of them having exactly one child of type α_i'. Hence, there are exactly v_s nodes of type α_i', being the last one of the form shown in Figure B.3 (see node r_4). By definition of $P(\alpha_i')$, $|\{x \mid x \text{ is a child of } r_4 \text{ of type } c_i\}| = |\{x \mid x \text{ is a child of } r_4 \text{ of type } e_i\}|$. Given that $r(n_t.@v \subseteq_{FK} e_i.@v)$, $r(e_i.@v \subseteq_{FK} n_t.@v)$ are in Σ and that every node of type e_i in T is a child of r_4, $|\{x \mid x \text{ is a child of } r_4 \text{ of type } c_i\}| = |ext(n_t)|$. Thus, since r_4

is a node of type α'_i and $\alpha'_i(\beta_i.@v \subseteq_{FK} c_i.@v)$, $\alpha'_i(c_i.@v \subseteq_{FK} \beta_i.@v)$ are in Σ, $|\{x \mid x$ is a child of r_4 of type $\beta_i\}| = |ext(n_t)| = v_t$. In addition, by definition of $P(\alpha'_i)$, the number of children of r_4 of type d_i is $2v_t$.

Given that r_3 is a node of type α_i and $\alpha_i(\beta_i.@v \subseteq_{FK} d_i.@v)$, $\alpha_i(d_i.@v \subseteq_{FK} \beta_i.@v)$ are in Σ, $|\{x \mid x$ is a child of r_3 of type $\beta_i\}| = v_t$, since there are $2v_t$ descendants of r_3 of type d_i and v_t children of r_4 of type β_i. Furthermore, by definition of $P(\alpha_i)$, the number of children of r_3 of type X is $a_i v_t$ and the number of children of r_3 of type c_i is $2v_t$. We can use the same argument to prove that the number of children of r_2 of types β_i and d_i are v_t and $2v_t$, respectively. Thus, the number of children of r_1 of type X is $a_i v_t$ and the number of descendants of r_1 of type X is $2a_i v_t$. If we continue with this process we can prove, by induction, that the number of X-nodes in T that are children of some node of type α_i is $v_s a_i v_t$, since there are v_s nodes of type α_i in T. This conclude the proof, since $|\{x \mid x$ is an X-node in T and x is a child of a node of type $\alpha_i\}| = a_i v_s v_t$.

B.4 Proof of Theorem 5.5.7

We will reduce SAT-CNF to our problem. Let φ be a propositional formula $C_1 \wedge \cdots \wedge C_n$, where each C_i is a clause. Assume that each C_i ($i \in [1, n]$) contains neither repeated nor complementary literals and φ mentions propositional variables $x_1, \ldots, x_m$.

We will define a non-recursive no-star DTD D and a set of unary keys Σ such that φ is satisfiable iff (D, Σ) is consistent. Define $D = (E, A, P, R, r)$ as follows.

- $E = \{r\} \cup \{X_{i,j} \mid C_i$ contains literal $x_j\} \cup \{\bar{X}_{i,j} \mid C_i$ contains literal $\neg x_j\}$.
- If $C_1 = \bigvee_{k=1}^{p} x_{i_k} \vee \bigvee_{k=1}^{q} \neg x_{j_k}$, then $P(r) = X_{1,i_1}|\cdots|X_{1,i_p}|\bar{X}_{1,j_1}|\cdots|\bar{X}_{1,j_q}$. For each $l \in [2, n]$, if $C_l = \bigvee_{k=1}^{p} x_{i_k} \vee \bigvee_{k=1}^{q} \neg x_{j_k}$, then for each $X_{l-1,j} \in E$, $P(X_{l-1,j}) = X_{l,i_1}|\cdots|X_{l,i_p}|\bar{X}_{l,j_1}|\cdots|\bar{X}_{l,j_q}$, and for each $\bar{X}_{l-1,j} \in E$, $P(\bar{X}_{l-1,j}) = X_{l,i_1}|\cdots|X_{l,i_p}|\bar{X}_{l,j_1}|\cdots|\bar{X}_{l,j_q}$.
- $A = \{@l_{i,j,k} \mid i < j$ and $x_k, \neg x_k$ are contained in the union of the literals of C_i and $C_j\}$.
- For each $X_{i,j} \in E$, $R(X_{i,j}) = \{@l_{k,i,j'} \mid j \neq j'$ and $@l_{k,i,j'} \in A\}$. For each $\bar{X}_{i,j} \in E$, $R(\bar{X}_{i,j}) = \{@l_{k,i,j'} \mid j \neq j'$ and $@l_{k,i,j'} \in A\}$. Furthermore, $R(r) = \emptyset$.

For example, if $\varphi = (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$, then D is equal to the DTD shown in figure B.4

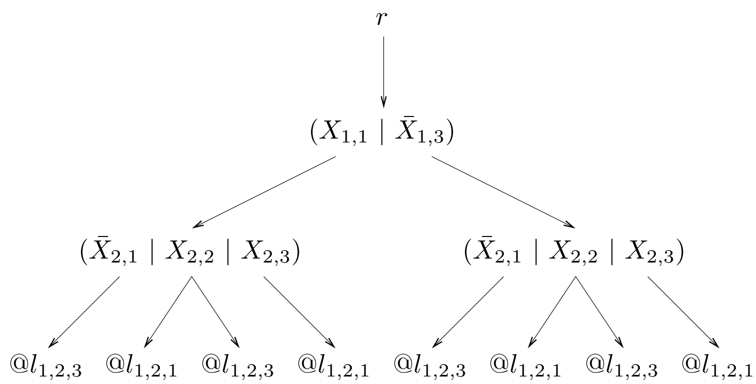


Figure B.4: DTD generated from $(x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3)$.

A set of unary keys Σ is defined as follows. If $X_{i,k} \in E$, $\bar{X}_{j,k} \in E$ and $i < j$, then $r//X_{i,k}[//@l_{i,j,k}] \to r//X_{i,k}$ is in Σ. If $\bar{X}_{i,k} \in E$, $X_{j,k} \in E$ and $i < j$, then $r//\bar{X}_{i,k}[//@l_{i,j,k}] \to r//\bar{X}_{i,k}$ is in Σ.

XML trees conforming to D represent truth assignments for variables $x_1, \ldots, x_m$. For example, one XML tree conforming to the DTD shown in figure B.4 contains nodes of types r, $\bar{X}_{1,3}$, $X_{2,2}$, and attributes $@l_{1,2,1}$ and $@l_{1,2,3}$ in the node of type $X_{2,2}$. This tree represents the truth assignment $\sigma(x_3) = 0$ and $\sigma(x_2) = 1$. We use the set of keys Σ to avoid inconsistent assignments. For instance, there is an XML tree T conforming to the DTD shown in figure B.4 containing nodes of types r, $X_{1,1}$, $\bar{X}_{2,1}$, and attribute $@l_{1,2,3}$ in the node of type $\bar{X}_{2,1}$. T cannot represent a truth assignment since it says that 0 and 1 must be assigned to x_1. T does not satisfy Σ, since $r//X_{1,1}[//@l_{1,2,1}] \to r//X_{1,1} \in \Sigma$ and, therefore, T must contain a node of type either $X_{2,2}$ or $X_{2,3}$.

We have to prove that φ is satisfiable iff (D, Σ) is consistent. Here, we will prove only the "if" direction. The "only if" direction is similar.

Assume that $T = (V, \mathit{lab}, \mathit{ele}, \mathit{att}, \mathit{root})$ is an XML tree conforming to D and satisfying Σ. Define a truth assignment σ as follows. For each variable x_j ($j \in [1, m]$), if there is a node in V of type $X_{i,j}$ ($i \in [1, n]$), then $\sigma(x_j) = 1$, otherwise, $\sigma(x_j) = 0$. We have to prove that σ satisfies φ. Let C_i be a clause in φ ($i \in [1, n]$). By definition of D, there is a literal x_j ($j \in [1, m]$) in C_i and a node in V of type $X_{i,j}$ or there is a literal $\neg x_k$ ($k \in [1, m]$) and a node in V of type $\bar{X}_{i,k}$. In the former case, $\sigma(x_j) = 1$ and, therefore, σ satisfies C_i since x_j is a literal in this clause. In the latter case, assume that there is a node in V of type $\bar{X}_{i,k}$ ($k \in [1, m]$). If $\sigma(x_k) = 1$, then there is a node in V of type $X_{i',k}$ ($i' \in [1, n]$). We have to consider two cases.

1. If $i' < i$, then $r//X_{i',k}[//@l_{i',i,k}] \rightarrow r//X_{i',k} \in \Sigma$. By definition of R, $@l_{i',i,k} \notin R(\bar{X}_{i,k})$. Thus, $T \not\models \Sigma$ since there is a node x in T reachable from the root by following a path in $r//X_{i',k}$ such that $reach(x, //@l_{i',i,k}) = \emptyset$.

2. If $i' > i$, then $r//\bar{X}_{i,k}[//@l_{i,i',k}] \rightarrow r//\bar{X}_{i,k} \in \Sigma$. By definition of R, $@l_{i,i',k} \notin R(X_{i',k})$. Thus, $T \not\models \Sigma$ since there is a node x in T reachable from the root by following a path in $r//\bar{X}_{i,k}$ such that $reach(x, //@l_{i,i',k}) = \emptyset$.

In both cases we reach a contradiction since we assume that $T \models \Sigma$. Thus, we conclude that V does not contain a node of type $X_{i',k}$ ($i' \in [1, n]$) and, therefore, $\sigma(x_k) = 0$. Hence, σ satisfies C_i since $\neg x_k$ is a literal in this clause. This concludes the proof of the theorem.

Appendix C

Proofs from Chapter 6

A DTD D can be inconsistent in the sense that there is no XML tree T such that $T \models D$. For example, a recursive DTD containing a rule $P(a) = a$ is not consistent; there is no a finite XML tree satisfying this rule. In this section we only consider consistent DTDs, since the implication problem for inconsistent DTDs is trivial and it can be checked in linear time whether a DTD is consistent [FL01].

C.1 Proof of Theorem 6.3.1

To prove this theorem, we need to introduce some terminology and prove two technical lemmas. Given an XML tree $T = (V, \mathit{lab}, \mathit{ele}, \mathit{att}, \mathit{root})$ and a node $v \in V - \{\mathit{root}\}$, define T_{-v} as an XML tree constructed by removing v and all the descendant of v from T. Formally, T_{-v} is defined to be $(V', \mathit{lab}', \mathit{ele}', \mathit{att}', \mathit{root})$, where $V' = V - \{x \in V \mid x = v$ or x is a descendant of $v\}$, $\mathit{lab}' = \mathit{lab}\restriction_{V'}$, $\mathit{att}' = \mathit{att}\restriction_{V' \times \mathit{Att}}$ and ele' is defined as follows. For every $v' \in V'$ such that v' is not the parent of v in T, $\mathit{ele}'(v') = \mathit{ele}(v')$. For the parent $v^\star$ of v in T, assume that $\mathit{ele}(v^\star) = [v_1, \ldots, v_n]$ and that v is the i-th child of $v^\star$, and define $\mathit{ele}'(v^\star)$ as $[v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n]$.

Given an XML tree T, if v is a node of T having a sibling v' of the same type ($\mathit{lab}(v) = \mathit{lab}(v')$), then v is said to be *removable from* T. The following lemma follows from the definitions of T_{-v} and tuples_D.

Lemma C.1.1 *Given a DTD D, a tree $T \lhd D$ and a node v of T, if v is removable from T, then $\mathit{tuples}_D(T_{-v}) \subsetneqq \mathit{tuples}_D(T)$.*

Given a regular expression β, denote by $L(\beta)$ the language defined by β. There are well-known polynomial time algorithms that given a regular expression β, generate nondeterministic automata accepting the same language as β. Choose one of these algorithms, and assume that its running time is $O(|\beta|^c)$, where c is a fixed constant. Furthermore, define $\mathcal{A}_\beta$ as the automaton generated by the algorithm on input β, and let Q_β be the set of states of $\mathcal{A}_\beta$.

Given strings w and w' over the same alphabet, we say that w' is contained in w if there exist $k \geq 1$ and strings $u_1, \ldots, u_k, v_1, \ldots, v_{k-1}$ such that $w = u_1 v_1 \cdots u_{k-1} v_{k-1} u_k$ and $w' = u_1 \cdots u_{k-1} u_k$. Thus, w' is contained in w if w' can be obtained by removing from w some of its substrings.

Lemma C.1.2 *Let β be a regular expression, $w \in L(\beta)$ and Γ the set of symbols mentioned in w. Then there exists a string $w' \in L(\beta)$ such that:*

1. *w' is contained in w and the set of symbols mentioned in w' is Γ;*
2. *$|w'| \leq 2 \cdot (|Q_\beta| + 1) \cdot (|\Gamma| + 1)$;*
3. *for every $a \in \Gamma$, if a appears more than once in w, then it also appears more than once in w'.*

Proof: If w is the empty string, then the property trivially holds. Thus, assume that $w = a_1 \cdots a_n$, where $n \geq 1$ and each $a_i \in \Gamma$ ($i \in [1, n]$).

Define a set $I \subseteq [1, n]$ as follows. For every $a \in \Gamma$, if a appears only once in w, then pick an arbitrary $i \in [1, n]$ such that $a_i = a$ and let i be an element of I. Otherwise, pick distinct $i, j \in [1, n]$ such that $a_i = a_j = a$ and let i, j be elements of I. Finally, let 1 and n be elements of I.

Given that $w \in L(\beta)$, there exists an accepting run of $\mathcal{A}_\beta$ on w, that is, a function $\rho : \{1, \ldots, n\} \to Q_\beta$ such that $\rho(1) \in \delta_\beta(q_0, a_1)$, $\rho(i+1) \in \delta_\beta(\rho(i), a_{i+1})$ ($i \in [1, n-1]$) and $\rho(n) \in F_\beta$, where q_0, δ_β and F_β are the initial state, the transition function and the set of final states of automaton $\mathcal{A}_\beta$, respectively. We use ρ to define string w'. More precisely, we can construct a string w' satisfying the conditions of the lemma by choosing some of the substrings u of w such that $u = a_i \cdots a_j$ ($1 \leq i < j \leq n$), $[i, j] \cap I = \emptyset$ and $\rho(i) = \rho(j)$, and then removing $a_{i+1} \cdots a_j$ from w. In particular, we have that $|w'| \leq |I| + |Q_\beta| \cdot (|I| - 1)$ since there exists an accepting run of $\mathcal{A}_\beta$ on w' that has no cycles in between two positions of this string that correspond to two consecutive positions in I. Therefore, given that $|I| \leq 2 \cdot |\Gamma| + 2$, we have that $|w'| \leq (2 \cdot |\Gamma| + 2) + |Q_\beta| \cdot (2 \cdot |\Gamma| + 1) \leq$

$(2 \cdot |\Gamma| + 2) + |Q_\beta| \cdot (2 \cdot |\Gamma| + 2) = (|Q_\beta| + 1) \cdot (2 \cdot |\Gamma| + 2) = 2 \cdot (|Q_\beta| + 1) \cdot (|\Gamma| + 1)$. This concludes the proof of the lemma. □

PROOF OF THEOREM 6.3.1: We show that the complement of our problem is in NEXPTIME. More precisely, we prove that given a DTD D and set of functional dependencies $\Sigma \cup \{\varphi\}$ over D, if $(D, \Sigma) \not\vdash \varphi$, then there exists an XML tree T such that $T \models (D, \Sigma)$, $T \not\models \varphi$ and $\|T\|$ is $\|D\|^{O(\|D\|+\|\Sigma\|)}$.

Let $D = (E, A, P, R, r)$ be a DTD and $\Sigma \cup \{\varphi\}$ a set of functional dependencies such that $(D, \Sigma) \not\vdash \varphi$. Assume that φ is of the form $S \to p$, where $S \cup \{p\} \subseteq \mathit{paths}(D)$. Given that $(D, \Sigma) \not\vdash \varphi$, there exists an XML tree T' conforming to D and satisfying Σ such that $T' \not\models \varphi$. Furthermore, given that $\Sigma \cup \{\varphi\}$ only mentions a finite number of paths, we assume that the depth of T' is at most $\|\Sigma\| + \|D\|$. We observe that the size of T' can be arbitrarily large.

Given that $T' \not\models \varphi$, there exists tree tuples $t_1, t_2 \in \mathit{tuples}_D(T')$ such that $t_1.q = t_2.q$ and $t_1.q \neq \bot$, for every $q \in S$, and $t_1.p \neq t_2.p$. By Lemmas C.1.1 and C.1.2, we know that there exists a subtree T of T' such that $T \models D$, $t_1, t_2 \in \mathit{tuples}_D(T)$, $\mathit{tuples}_D(T) \subseteq \mathit{tuples}_D(T')$ and for every element type $\tau \in E$ and every τ-node v of T, the number of children of v is at most $2 \cdot (|Q_{P(\tau)}| + 1) \cdot (|E| + 1)$. Thus, we have that for every τ-node v of T, the number of children of v is $O(\|D\|^{c+1})$ since $|Q_{P(\tau)}|$ is $O(|P(\tau)|^c)$, $|P(\tau)| \leq \|D\|$ and $|E| \leq \|D\|$. We conclude that $\|T\|$ is $\|D\|^{O(\|D\|+\|\Sigma\|)}$ since the depth of T is at most $\|D\| + \|\Sigma\|$. Furthermore, we deduce that $T \models \Sigma$, since $\mathit{tuples}_D(T) \subseteq \mathit{tuples}_D(T')$ and $T' \models \Sigma$, and that $T \not\models \varphi$, since $t_1, t_2 \in \mathit{tuples}_D(T)$. Hence, there exists an XML tree T such that $T \models (D, \Sigma)$, $T \not\models \varphi$ and $\|T\|$ is $\|D\|^{O(\|D\|+\|\Sigma\|)}$. This concludes the proof of the theorem. □

C.2 Proof of Theorem 6.3.2

To prove this theorem we start by introducing some terminology. Given a simple DTD $D = (E,\ A,\ P,\ R,\ r)$ and $p,\ p' \in \mathit{paths}(D)$ such that p is a proper prefix of p', we say that p' *can be nullified from* p if p' is of the form $p.w_1. \cdots .w_n$, where $w_i \in E \cup A \cup \{\mathtt{S}\}$ ($i \in [1, n]$) and either (1) $P(\mathit{last}(p))$ contains $w_1?$ or w_1^*; or (2) there is $i \in [1, n-1]$ such that $P(w_i)$ contains $w_{i+1}?$ or w_{i+1}^*. Intuitively, p' can be nullified from p if there exists and XML tree T conforming to D and a tree tuple t in T such that $t.p \neq \bot$ and $t.p' = \bot$. For example, if $P(r) = a$, $P(a) = b^*$ and $P(b) = c$, then $r.a.b.c$ can be nullified from r and $r.a$, but it cannot be nullified from $r.a.b$. Given $S \subseteq \mathit{paths}(D)$, we say that p' *can be* *nullified from* S if p' can be nullified from p, where p is the longest common prefix of p' and a path from S.

The following is proved by the same argument as Lemma C.4.1 shown in electronic appendix C.4.

Lemma C.2.1 *Given a simple DTD* D*, a set* Σ *of functional dependencies over* D *and* $S \cup \{p\} \subseteq \mathit{paths}(D)$*,* $(D, \Sigma) \not\vdash S \to p$ *if and only if there is an XML tree* T *and a path* q *prefix of* p *such that* $T \models (D, \Sigma)$*,* $\mathit{tuples}_D(T) = \{t_1, t_2\}$*,* $t_1.S = t_2.S$*,* $t_1.S \neq \bot$*,* $t_1.p \neq t_2.p$*,* $t_1.p \neq \bot$*,* $t_2.p \neq \bot$*,* $t_1.q \neq t_2.q$ *and*

- *For each* $s \in \mathit{paths}(D)$*, if* s *can be nullified from* $S \cup \{p\}$*, then* $t_1.s = t_2.s = \bot$*.*

- *For each* $s \in \mathit{paths}(D)$*, if* q *is not a prefix of* s *and* s *cannot be nullified from* $S \cup \{p\}$*, then* $t_1.s = t_2.s$ *and* $t_1.s \neq \bot$*.*

To prove that the implication problem for simple DTDs can be solved in polynomial time, we use the technique of [SDPF81] and code constraints with propositional formulas. That is, for each simple DTD D and set of functional dependencies $\Sigma \cup \{S \to p\}$ over D, we will define a propositional formula φ such that $(D, \Sigma) \not\vdash S \to p$ if and only if φ is satisfiable. This formula will be of the form $\varphi_1 \vee \cdots \vee \varphi_n$, where each φ_i ($i \in [1, n]$) is a conjunction of Horn clauses. Given that the consistency problem for Horn clauses is solvable in linear time, we will conclude that our problem is solvable in quadratic time.

Let D be a DTD, Σ a set of functional dependencies over D and $S \cup \{p\} \subseteq \mathit{paths}(D)$. Recall that we assumed that each constraints in Σ is of the form $S' \to p'$, where $S' \cup \{p'\} \subseteq \mathit{paths}(D)$. We define $\mathit{paths}(\Sigma)$ as $\{s \mid \text{there is } S' \to p' \in \Sigma \text{ such that } s \in S' \cup \{p'\}\}$. To define the propositional formula φ we view each path $s \in \mathit{paths}(\Sigma) \cup S \cup \{p\}$ as a propositional variable. Furthermore, for each path q which is a prefix of p we define a propositional formula φ_q as

$$\neg p \wedge (\bigwedge_{s \in P_q \cup S} s) \wedge (\bigwedge_{s \in N_q} \neg s) \wedge \bigwedge_{\psi \in \Gamma} \psi,$$

where P_q, N_q and Γ are set of propositional variables and formulas defined as follows.

- For each $s \in \mathit{paths}(\Sigma)$ such that s cannot be nullified from $S \cup \{p\}$ and q is not a prefix of s, s is included in P_q.

- For each $s \in \mathit{paths}(\Sigma)$ such that $s \in \mathit{EPaths}(D)$, s cannot be nullified from $S \cup \{p\}$ and q is a prefix of s, s is included in N_q.

- For each $S' \to p' \in \Sigma$, if there is no $q' \in S' \cup \{p'\}$ such that q' can be nullified from $S \cup \{p\}$, then $(\bigwedge_{s \in S'} s) \to p'$ is included in Γ

We note that φ_q is a conjunction of Horn clauses.

The propositional formula φ is defined as the disjunction of some of the formula φ_qs. The following lemma shows that in this disjunction we only need to consider qs such that $q = q'.\tau$, for some $\tau \in E$, and $P(last(q'))$ contains τ^* or τ^+.

Lemma C.2.2 *Let $D = (E, A, P, R, r)$ be a simple DTD, Σ a set of functional dependencies over D and $S \cup \{p, q\} \subseteq paths(D)$ such that q is a prefix of p. If there is $\tau \in E$ such that $q = q'.\tau$ and $P(last(q'))$ contains τ^* or τ^+, then φ_q is satisfiable iff there is an XML tree T such that $T \models (D, \Sigma)$, $tuples_D(T) = \{t_1, t_2\}$, $t_1.S = t_2.S$, $t_1.S \neq \perp$, $t_1.p \neq t_2.p$, $t_1.p \neq \perp$, $t_2.p \neq \perp$, $t_1.q \neq t_2.q$ and*

- *For each $s \in paths(D)$, if s can be nullified from $S \cup \{p\}$, then $t_1.s = t_2.s = \perp$.*
- *For each $s \in paths(D)$, if q is not a prefix of s and s cannot be nullified from $S \cup \{p\}$, then $t_1.s = t_2.s$ and $t_1.s \neq \perp$.*

Proof: ($\Rightarrow$) Let σ be a truth assignment satisfying φ_q. We define tuples t_1 and t_2 as follows. For each $s \in paths(D)$, if s can be nullified from $S \cup \{p\}$, then $t_1.s = t_2.s = \perp$. If s cannot be nullified from $S \cup \{p\}$ we consider two cases. If q is not a prefix of s, then $t_1.s = t_2.s$ and $t_1.s \neq \perp$. Otherwise, if $\sigma(s) = 1$, then $t_1.s = t_2.s$ and $t_1.s \neq \perp$, else $t_1.s \neq t_2.s$, $t_1.s \neq \perp$ and $t_2.s \neq \perp$.

It is straightforward to prove that there is an XML tree $T \in trees_D(\{t_1, t_2\})$ such that $T \models D$ and $tuples_D(T) = \{t_1, t_2\}$. Given that $\sigma \models \neg p \wedge \bigwedge_{s \in S} s$, $t_1.S = t_2.S$, $t_1.S \neq \perp$, $t_1.p \neq t_2.p$, $t_1.p \neq \perp$ and $t_2.p \neq \perp$. Besides, $t_1.q \neq t_2.q$, since $q \in N_q$ and $\sigma \models \bigwedge_{s \in N_q} \neg s$. Thus, to finish the proof we have to show that $T \models \Sigma$. Let $S' \to p' \in \Sigma$. If there is $q' \in S' \cup \{p'\}$ such that q' can be nullified from $S \cup \{p\}$, then T trivially satisfies $S' \to p'$ since $t_1.q' = t_2.q' = \perp$. Otherwise, suppose that $t_1.S' = t_2.S'$ and $t_1.S' \neq \perp$. Then, by considering that $\sigma \models \bigwedge_{s \in P_q} s$ and the definition of t_1 and t_2, we conclude that $\sigma \models \bigwedge_{s \in S'} s$. Thus, given that $\sigma \models (\bigwedge_{s \in S'} s) \to p'$, we conclude that $\sigma(p') = 1$, and, therefore, $t_1.p' = t_2.p'$.

($\Leftarrow$) Suppose that there is an XML tree T satisfying the conditions of the lemma. Define a truth assignment σ as follows. For each $s \in paths(\Sigma) \cup S \cup \{p\}$, if $t_1.s \neq t_2.s$ then $\sigma(s) = 0$. Otherwise, $\sigma(s) = 1$.

Given that $t_1.p \neq t_2.p$ and $t_1.S = t_2.S$, $\sigma(\neg p) = 1$ and $\sigma \models \bigwedge_{s \in S} s$. Let $s \in P_q$. By definition, s cannot be nullified from $S \cup \{p\}$ and q is not a prefix of s, and, therefore, $t_1.s = t_2.s$. Thus, $\sigma(s) = 1$. We conclude that $\sigma \models \bigwedge_{s \in P_q} s$. Let $s \in N_q$. By definition, s cannot be nullified from $S \cup \{p\}$, q is a prefix of s and $s \in \mathit{EPaths}(D)$. Hence, $t_1.s \neq t_2.s$ and $\sigma(s) = 0$. We conclude that $\sigma \models \bigwedge_{s \in N_q} \neg s$. Finally, let $(\bigwedge_{s \in S'} s) \to p' \in \Sigma_q$. If $\sigma \models \bigwedge_{s \in S'} s$, then by definition of σ and Σ_q, we conclude that $t_1.S' = t_2.S'$ and $t_1.S' \neq \perp$. Thus, given that $T \models \Sigma$, we conclude that $t_1.p' = t_2.p'$ and, therefore, $\sigma(p') = 1$. $\square$

Combining Lemmas C.2.1 and C.2.2 we obtain:

Lemma C.2.3 *Let $D = (E, A, P, R, r)$ be a simple DTD, Σ a set of functional dependencies over D and $S \cup \{p\} \subseteq \mathit{paths}(D)$. Assume that $X = \{q \in \mathit{paths}(D) \mid q$ is a prefix of p and there is $\tau \in E$ such that $q = q'.\tau$ and $P(\mathit{last}(q'))$ contains τ^* or $\tau^+\}$. Then, $(D, \Sigma) \not\vdash S \to p$ iff $\varphi = \bigvee_{q \in X} \varphi_q$ is satisfiable.*

Finally, we are ready to show that for a simple DTD D and a set of FDs $\Sigma \cup \{S \to p\}$ over D, checking whether $(D, \Sigma) \vdash S \to p$ can be done in quadratic time. The size of each formula φ_q in the previous Lemma is $O(\|\Sigma\| + \|S\| + \|p\|)$. Thus, it is possible to verify whether φ_q is satisfiable in time $O(\|\Sigma\| + \|S\| + \|p\|)$, since satisfiability of propositional Horn formulas can be checked in linear time [DG84]. Hence, given that there are at most $\|p\|$ of these formulas, checking whether formula $\bigvee_{q \in X} \varphi_q$ in Lemma C.2.3 is satisfiable requires time $O(\|p\| \cdot (\|\Sigma\| + \|S\| + \|p\|))$. To construct this formula, first we execute two steps:

1. For every $s \in \mathit{paths}(\Sigma)$, find the longest common prefix of s and a path from $S \cup \{p\}$, which requires time $O(\|s\| \cdot (\|S\| + \|p\|))$. By using this prefix verify whether s can be nullified from $S \cup \{p\}$, which requires time $O(\|s\| \cdot \|D\|)$.

2. For each $s \in \mathit{paths}(\Sigma)$ and for each prefix q of p, verify whether q is a prefix of s, which requires time $O(\|q\|)$.

The total time required by these steps is $O(\|\Sigma\| \cdot (\|D\| + \|S\| + \|p\|))$. Let k be the number of paths in Σ and l be the number of prefixes of p. The information generated by the first step is stored in a array with k entries, one for each path in Σ, indicating whether each of these paths can be nullified from $S \cup \{p\}$. Similarly, the information generated by the second step is stored in l arrays with k entries each. By using these data structures, the formula $\bigvee_{q \in X} \varphi_q$ in Lemma C.2.3 can be constructed in time $O(\|p\| \cdot (\|\Sigma\| + \|S\| + \|p\|))$.

Thus, the total time of the algorithm is $O(\|p\| \cdot (\|\Sigma\| + \|S\| + \|p\|) + \|\Sigma\| \cdot (\|D\| + \|S\| + \|p\|))$. This completes the proof of Theorem 6.3.2.

C.3 Proof of Theorem 6.3.3

To prove this theorem first we prove two lemmas. Let $D = (E, A, P, R, r)$ be a disjunctive DTD and $\tau \in E$ such that $P(\tau) = s_1, \ldots, s_n$. Assume that for a fixed $k \in [1, n]$, $s_k = s'_1 | s'_2$, where s'_1, s'_2 are simple disjunctions over alphabets A'_1, A'_2 and $A'_1 \cap A'_2 = \emptyset$. Assume that there is only one $p_\tau \in \mathit{paths}(D)$ such that $\mathit{last}(p_\tau) = \tau$. We define $\mathit{paths}_i(D)$ (for $i = 1, 2$) as the set of all paths q in D such that one of the following statement holds: (1) p_τ is not a proper prefix of q or (2) there is $\tau' \in E$ such that $p_\tau.\tau'$ is a prefix of q and τ' is in the alphabet of any of the regular expressions s_1, $\ldots$, s_{k-1}, s'_i, s_{k+1}, $\ldots$, s_n. Then we define DTDs $D_i = (E_i, A_i, P_i, R_i, r)$ (for $i = 1, 2$) as follows. $E_i = \{\tau' \in E \mid \tau'$ is mentioned in some $q \in \mathit{paths}_i(D)\}$, $A_i = \{@l \mid$ there is $\tau' \in E_i$ such that $@l \in R(\tau')\}$, $P_i(\tau) = s_1, \ldots, s_{k-1}, s'_i, s_{k+1}, \ldots, s_n$, $P_i(\tau') = P(\tau')$, for each $\tau' \in E_i - \{\tau\}$, and $R_i = R|_{E_i}$. Moreover, given a set of functional dependencies Σ over D, we define a set of functional dependencies Σ_i over D_i (for $i = 1, 2$) as follows. For each $S \to p \in \Sigma$, if $S \cup \{p\} \subseteq \mathit{paths}_i(D)$, then $S \to p$ is included in Σ_i.

Lemma C.3.1 *Let D, Σ, τ, p_τ, D_i and Σ_i, for $i = 1, 2$ be as above and let $S \to p$ be a functional dependency over D. Then*

(a) If $S \cup \{p\} \not\subseteq \mathit{paths}_i(D)$ for every $i \in [1, 2]$, then $(D, \Sigma) \vdash S \to p$.

(b) If $S \cup \{p\} \subseteq \mathit{paths}_1(D)$ and $S \cup \{p\} \not\subseteq \mathit{paths}_2(D)$, then $(D, \Sigma) \vdash S \to p$ iff $(D_1, \Sigma_1) \vdash S \to p$.

(c) If $S \cup \{p\} \subseteq \mathit{paths}_i(D)$ for every $i \in [1, 2]$, then $(D, \Sigma) \vdash S \to p$ iff for every $i \in [1, 2]$, $(D_i, \Sigma_i) \vdash S \to p$.

PROOF: (a) Let $p_i \in \mathit{paths}_i(D)$ ($i \in [1, 2]$) such that $p_i \in S \cup \{p\}$, for every $i \in [1, 2]$, $p_1 \notin \mathit{paths}_2(D)$ and $p_2 \notin \mathit{paths}_1(D)$. Let T be an XML tree such that $T \models (D, \Sigma)$, and $t_1, t_2 \in \mathit{tuples}_D(T)$. Without loss of generality, assume that $p_1 \in S$. If $t_1.p_1 = t_2.p_1$ and $t_1.p_1 \neq \bot$, then $t_1.p_2 = t_2.p_2 = \bot$, and, therefore, $T \models S \to p$. Thus, we conclude that $(D, \Sigma) \vdash S \to p$.

(b) If $(D, \Sigma) \vdash S \rightarrow p$, we have to prove that $(D_1, \Sigma_1) \vdash S \rightarrow p$. Let T_1 be an XML such that $T_1 \models (D_1, \Sigma_1)$. This tree conforms to D and satisfies Σ, since each constraint $\varphi \in \Sigma - \Sigma_1$ contains at least one path q such that for every $t \in \mathit{tuples}_D(T_1)$, $t.q = \perp$. Hence, $T_1 \models S \rightarrow p$.

Suppose that $(D_1, \Sigma_1) \vdash S \rightarrow p$. We have to prove that $(D, \Sigma) \vdash S \rightarrow p$. Let T be an XML tree such that $T \models (D, \Sigma)$, and $t_1, t_2 \in \mathit{tuples}_D(T)$. Let $p_1 \in \mathit{paths}_1(D)$ such that $p_1 \in S \cup \{p\}$ and $p_1 \notin \mathit{paths}_2(D)$. By contradiction, suppose that $t_1.S = t_2.S$, $t_1.S \neq \perp$ and $t_1.p \neq t_2.p$. If $p_1 \in S$, then there is $T_1 \in \mathit{trees}_D(\{t_1, t_2\})$ such that $T_1 \models D_1$, since $t_1.p_1 \neq \perp$ and $t_2.p_1 \neq \perp$. Since $T \models \Sigma$, $T_1 \models \Sigma_1$, and, therefore $(D_1, \Sigma_1) \nvdash S \rightarrow p$, a contradiction. If $p_1 = p$, without loss of generality, we can assume that $t_1.p_1 \neq \perp$. If $t_2.p_1 \neq \perp$, then there is $T_1 \in \mathit{trees}_D(\{t_1, t_2\})$ such that $T_1 \models D_1$. But, $T_1 \models \Sigma_1$, since $T \models \Sigma$, and, therefore $(D_1, \Sigma_1) \nvdash S \rightarrow p$, a contradiction. Assume that $t_2.p_1 = \perp$. Define $t_2' \in \mathcal{T}(D_1)$ as follows. For each $w \in \mathit{paths}_1(D) \cap \mathit{paths}_2(D)$, $t_2'.w = t_2.w$, and for each $w \in \mathit{paths}_1(D) - \mathit{paths}_2(D)$, if $t_1.w = \perp$, then $t_2'.w = \perp$, otherwise $t_2'.w \neq t_1.w$. Given that $t_1.p_\tau \neq t_2.p_\tau$, since $t_1.p_1 \neq \perp$ and $t_2.p_1 = \perp$, we conclude that there is an XML tree $T_1 \in \mathit{trees}_D(\{t_1, t_2'\})$ such that T_1 conforms to D_1. But $T_1 \models \Sigma_1$, since $\mathit{trees}_D(\{t_1, t_2\}) \models \Sigma$. Thus, $(D_1, \Sigma_1) \nvdash S \rightarrow p$, again a contradiction.

(c) We will only prove the "if" direction. The "only if" direction is analogous to the proof of this direction in (b). Assume that $(D, \Sigma) \nvdash S \rightarrow p$. We will show that $(D_1, \Sigma_1) \nvdash S \rightarrow p$ or $(D_2, \Sigma_2) \nvdash S \rightarrow p$.

Given that every disjunctive DTD is a relational DTD (see Proposition 6.3.4), by Lemma C.4.1 we conclude that $(D, \Sigma) \nvdash S \rightarrow p$ if and only if there is an XML tree T and a path q prefix of p such that $T \models (D, \Sigma)$, $\mathit{tuples}_D(T) = \{t_1, t_2\}$, $t_1.S = t_2.S$, $t_1.S \neq \perp$, $t_1.p \neq t_2.p$, $t_1.q \neq t_2.q$ and for each $s \in \mathit{paths}(D)$, if q is not a prefix of s, then $t_1.s = t_2.s$. We consider three cases.

1. If q is not a prefix of p_τ. Then, there is $T' \in \mathit{trees}_D(\{t_1, t_2\})$ such that T' conforms to either D_1 or D_2. Without loss of generality, assume that $T' \models D_1$. In this case, $T' \models \Sigma_1$, since $T \models \Sigma$. Hence, $(D_1, \Sigma_1) \nvdash S \rightarrow p$.

2. If q is a prefix of p_τ and there exists $a_1' \in A_1'$ and $a_2' \in A_2'$ such that $t_1.p_\tau.a_1' \neq \perp$ and $t_2.p_\tau.a_2' \neq \perp$. In this case, we define $t_2' \in \mathcal{T}(D_1)$ as follows. For each $w \in \mathit{paths}_1(D) \cap \mathit{paths}_2(D)$, $t_2'.w = t_2.w$, and for each $w \in \mathit{paths}_1(D) - \mathit{paths}_2(D)$, if $t_1.w = \perp$, then $t_2'.w = \perp$, otherwise $t_2'.w \neq t_1.w$. Then, there exists $T' \in$

$trees_{D_1}(\{t_1, t_2'\})$ such that $T' \models D_1$, $T' \models \Sigma_1$ and $T' \not\models S \to p$, since $T \models \Sigma$ and $T \not\models S \to p$. We conclude that $(D_1, \Sigma_1) \nvdash S \to p$.

3. If q is a prefix of p_τ and there are no $a_1' \in A_1'$ and $a_2' \in A_2'$ such that either $t_1.p_\tau.a_1' \neq \perp$ and $t_2.p_\tau.a_2' \neq \perp$ or $t_2.p_\tau.a_1' \neq \perp$ and $t_1.p_\tau.a_2' \neq \perp$. This case is analogous to the first one.

□

Given a disjunctive DTD $D = (E, A, P, R, r)$, to apply the previous lemma we need to find an element type τ such that there is exactly one path in D whose last element is τ and $P(\tau) = s_1, \ldots, s_k, \ldots, s_n$, where $s_k = s_1'|s_2'$, s_1' and s_2' are simple disjunctions over alphabets A_1', A_2' and $A_1' \cap A_2' = \emptyset$. If there is no such an element type and D is not a simple DTD, it is possible to create it by using the following transformation. Pick τ satisfying the previous conditions except for there is more than one path whose last element is τ. Pick $p \in paths(D)$ such that $last(p) = \tau$. Define a DTD $D_p = (E_p, A, P_p, R_p, r_p)$ as follows. $r_p = [r]$ and $E_p = (E - \{r\}) \cup \{[q] \mid q \in paths(D)$ and q is a prefix of $p\}$ (we use square brackets to distinguish between paths and element types). The functions P_p and R_p are defined as follows.

- For each $q \in paths(D)$ and $\tau' \in E$ such that $q.\tau'$ is a prefix of p, $P_p([q]) = f(P(last(q)))$, where f is a homomorphism defined as $f(\tau') = [q.\tau']$ and $f(\tau'') = \tau''$ for each $\tau'' \neq \tau'$. Moreover, $P_p([p]) = P(last(p))$ and $P_p(\tau') = P(\tau')$, for each $\tau' \in E - \{r\}$.
- For each $[q] \in E_p$, $R_p([q]) = R(last(q))$. Moreover, $R_p(\tau') = R(\tau')$, for each $\tau' \in E - \{r\}$.

Let $\Sigma \cup \{S \to q\}$ be a set of functional dependencies over D. We define a set of functional dependencies $\Sigma_p \cup \{S_p \to q_p\}$ over D_p as follows. For each path q' mentioned in $\Sigma \cup \{S \to q\}$, if $q' = q_1.q_2$, where q_1 is the longest common prefix of q' and p, then q' is replaced by $g(q_1).q_2$, where g is an homomorphism defined as $g([r]) = [r]$ and $g([w.\tau']) = g([w]).[w.\tau']$, for each $w.\tau'$ prefix of p. The following is straightforward.

Lemma C.3.2 *Let D, $\Sigma \cup \{S \to q\}$, D_p and $\Sigma_p \cup \{S_p \to q_p\}$ be as above. Then, $(D, \Sigma) \vdash S \to q$ iff $(D_p, \Sigma_p) \vdash S_p \to q_p$.*

Theorem 6.3.3 now follows from Lemmas C.3.1 and C.3.2.

C.4 The Implication Problem for Relational DTDs is in coNP

To prove this theorem we start with the following lemma.

Lemma C.4.1 *Given a relational DTD D, a set Σ of functional dependencies over D and $S \cup \{p\} \subseteq paths(D)$, $(D, \Sigma) \not\vdash S \to p$ if and only if there is an XML tree T and a path q prefix of p such that T conforms to D, T satisfies Σ, $tuples_D(T) = \{t_1, t_2\}$, $t_1.S = t_2.S$, $t_1.S \neq \perp$, $t_1.p \neq t_2.p$, $t_1.q \neq t_2.q$ and for each $s \in paths(D)$, if q is not a prefix of s, then $t_1.s = t_2.s$.*

Proof: We will prove only the "only if" direction, since the "if" direction is trivial.

Suppose that $(D, \Sigma) \not\vdash S \to p$. There is an XML tree T' conforming to D and satisfying Σ such that $T' \not\models S \to p$. Then, there are tuples $t'_1, t'_2 \in tuples_D(T)$ such that $t'_1.S = t'_2.S$, $t'_1.S \neq \perp$ and $t'_1.p \neq t'_2.p$. Let q be the shortest prefix of p such that $t'_1.q \neq t'_2.q$. We define tree tuples t_1 and t_2 as follows. For each $s \in paths(D)$, if q is not a prefix of s, then $t_1.s = t'_1.s$ and $t_2.s = t'_1.s$. Otherwise, $t_1.s = t'_1.s$ and $t_2.s = t'_2.s$. Notice that $t_1, t_2 \in tuples_D(T')$.

Given that D is a relational DTD, it is possible to find $T \in trees_D(\{t_1, t_2\})$ such that $T \models D$. We need to prove that T satisfies the conditions of the lemma. By definition of t_1 and t_2, $tuples_D(T) = \{t_1, t_2\}$ and for each $s \in paths(D)$, if q is not a prefix of s, then $t_1.s = t_2.s$. Besides, $t_1.S = t_2.S$, $t_1.S \neq \perp$ and $t_1.p \neq t_2.p$, since $t'_1.S = t'_2.S$, $t'_1.S \neq \perp$, $t'_1.p \neq t'_2.p$ and q is a prefix of p. Finally, $t_1.q \neq t_2.q$, since $t'_1.q \neq t'_2.q$, and $T \models \Sigma$, since $T' \models \Sigma$ and $t_1, t_2 \in tuples_D(T')$. □

Now we are ready to prove that the implication problem for relational DTDs is in coNP. Let D be a relational DTD, Σ a set of functional dependencies over D and $S \cup \{p\} \subseteq paths(D)$. Let $prefix(\Sigma \cup \{S \to p\})$ be the set of all $p' \in paths(D)$ such that p' is a prefix of a path mentioned in $\Sigma \cup \{S \to p\}$. Notice that $\|prefix(\Sigma \cup \{S \to p\})\|$ is $O(\|\Sigma \cup \{S \to p\}\|^2)$.

To check whether $(D, \Sigma) \not\vdash S \to p$, we use a nondeterministic algorithm that guesses the tuples t_1 and t_2 mentioned in Lemma C.4.1. This algorithm does not construct all the values in t_1 and t_2, it guesses only the values of these tuples that are necessary to verify whether $trees_D(\{t_1, t_2\}) \models \Sigma$. The algorithm works as follows. For each $s \in prefix(\Sigma \cup \{S \to p\})$, guess the values of $t_1.s$ and $t_2.s$. Verify whether it is possible to construct an XML tree conforming to D and containing t_1 and t_2. If this does not hold, then return “no”. Otherwise, guess a prefix q of p. Verify whether $t_1.S = t_2.S$, $t_1.S \neq \perp$, $t_1.p \neq t_2.p$, $t_1.q \neq t_2.q$ and for each $s \in \mathit{paths}(\Sigma \cup \{S \rightarrow p\})$, if q is not a prefix of s, then $t_1.s = t_2.s$. If this does not hold, then return “no”. Otherwise, check whether the values in t_1 and t_2 satisfy Σ. If this is the case, then return “yes”, otherwise return “no”.

The previous algorithm works in nondeterministic polynomial time, since $\|\mathit{prefix}(\Sigma \cup \{S \rightarrow p\})\|$ is $O(\|\Sigma \cup \{S \rightarrow p\}\|^2)$. Therefore, we conclude that the implication problem for relational DTDs is in coNP.