On Detecting and Repairing Inconsistent Schema Mappings

by

Terence Cheung-Fai Ho

B.Sc., The University of British Columbia, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University of British Columbia

(Vancouver)

October 2008

© Terence Cheung-Fai Ho, 2008
Abstract

Huge amount of data flows around the Internet every second, but for the data to be useful at its destination, it must be presented in a way such that the target has little problem interpreting it. Current data exchange technologies may rearrange the structure of data to suit expectations at the target. However, there may be semantics behind data (e.g. knowing the title of a book can determine its #pages) that may be violated after data translation. These semantics are expressed as integrity constraints (IC) in a database. Currently, there is no guarantee that the exchanged data conforms to the target's ICs. As a result, existing applications (e.g. user queries) that assume such semantics will no longer function correctly. Current constraint repair techniques deal with data after it has been translated; thus take no consideration of the integrity constraints at the source. Moreover, such constraint repair methods usually involve addition/deletion/modification of data, which may yield incomplete or false data. We consider the constraints of both source and target schemas; together with the mapping, we can efficiently detect which constraint is violated and suggest ways to correct the mappings.
# Table of Contents

Abstract ................................................................. ii

Table of Contents ....................................................... iii

List of Tables ............................................................ vi

List of Figures ........................................................... vii

1 Introduction ............................................................ 1

2 Definitions ............................................................. 10
  2.1 DTD definition .................................................. 10
  2.2 Simple DTD definition .......................................... 11
    2.2.1 Choice-free .............................................. 12
    2.2.2 Acyclic .................................................. 12
    2.2.3 Duplicate-free ........................................... 13
    2.2.4 Unorderness .............................................. 13
  2.3 DTD Group ....................................................... 14
    2.3.1 Single Path ............................................. 14
    2.3.2 DTD Group .............................................. 15
  2.4 XML Database .................................................. 16
    2.4.1 Simplifying text nodes to attributes ................ 16
    2.4.2 Notations .............................................. 16
  2.5 Missing Values and NULL Values .............................. 17
  2.6 XML constraint definitions ................................. 17
    2.6.1 Functional dependency ............................... 17

iii
2.6 XML constraint definitions ........................................... 17
  2.6.1 Functional dependency ......................................... 17
  2.6.2 Inclusion dependency .......................................... 18
2.7 Schema Mapping from HepToX ........................................... 18
2.8 Relational Constraint Definitions ................................... 20
  2.8.1 Relational Functional Dependency .............................. 20
  2.8.2 Relational Inclusion Dependency ............................... 20
  2.8.3 Typed Relational IND .......................................... 21
2.9 Typed XML Inclusion Dependency ................................. 21
2.10 Guaranteeing No Interaction of relational FD and IND Implication 22
  2.10.1 Proper Circular ............................................... 22
  2.10.2 Split-freeness ................................................. 23
  2.10.3 Intersection ................................................ 23
  2.10.4 Reduced set of FDs and INDs ................................. 24
  2.10.5 Theorem that Guarantees Non-Interaction between FD and IND ........................................... 24
3 Efficient XML Constraint Implication ............................... 25
  3.1 Reduction of XML constraint implication to relational constraint implication ........................................... 25
    3.1.1 Assumptions .................................................. 27
    3.1.2 XML Database Equivalence (≡) ................................ 27
    3.1.3 Schema Transformation Function — \( \mu(\Delta) \) ................. 29
    3.1.4 Constraint Transformation Function — \( \mu(\sigma_\Delta) \) ............... 30
    3.1.5 Data Transformation Functions — \( \mu(D_\Delta), \mu^{-1}(D_R) \) .......... 32
    3.1.6 Correctness of XML Constraint Propagation ......................... 35
    3.1.7 XML to Relational Constraint Implication Reduction .................. 38
    3.1.8 Algorithm to Test XML Constraint implication by Relational Constraint Implication ........................................... 39
  3.2 Guaranteeing No Interaction Between XML FDs and INDs .......... 40
    3.2.1 Retaining Proper Circular Property ............................ 40
    3.2.2 Retaining Reduced Property .................................. 43
    3.2.3 Inability to Retain Intersection Property ...................... 44
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>$\Sigma_{UBCBookstore}$</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>$\Sigma_{UBCLibrary}$</td>
<td>3</td>
</tr>
<tr>
<td>3.1</td>
<td>$\Psi(D_{UBCBookstore})$</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>$\mu(\Sigma_{UBCBookstore})$</td>
<td>33</td>
</tr>
<tr>
<td>3.3</td>
<td>$\mu(D_{UBCBookstore})$</td>
<td>34</td>
</tr>
</tbody>
</table>
List of Figures

1.1 $\Delta_{UBCBookstore}$ to $\Delta_{UBCLibrary}$ Example ............................................. 2
1.2 UBCBookstore Instance Example ................................................................. 5
1.3 Translated UBCLibrary Instance Example ......................................................... 5
1.4 Mapping Rules M for $UBCBookstore \rightarrow UBCLibrary$ Example ................. 6

2.1 $\Delta_{UBCBookstore}$ ................................................................. 15

3.1 Equivalent UBCBookstore Instance Example ................................................. 28
3.2 Schema Transformation Algorithm — $\mu(\Delta)$ .............................................. 29
3.3 Creating Transformation Generated Constraints Algorithm — $\Psi(\Delta)$ ........ 30
3.4 FD transformation Algorithm — $\mu(FD)$ .......................................................... 31
3.5 IND transformation Algorithm — $\mu(IND)$ ...................................................... 31
3.6 Constraints Transformation Algorithm — $\mu(\Sigma)$ ........................................ 32
3.7 Data Transformation Algorithm ................................................................. 33
3.8 Reverse Data Transformation Algorithm ....................................................... 35
3.9 Algorithm Testing XML Constraint Implication ............................................. 39
3.10 Example Showing Circular INDs after Transformation .................................. 41

4.1 FD Translation Algorithm — $\delta(FS,M)$ ......................................................... 50
4.2 IND Translation Algorithm — $\delta(IS,M)$ ....................................................... 51
4.3 Testing FD Propagation Algorithm .............................................................. 55
4.4 Testing IND Propagation Algorithm ............................................................ 57

5.1 FD violation Checking Algorithm .............................................................. 60
5.2 IND violation Checking Algorithm .............................................................. 62
# Glossary

$D_R$  
a relational database, 20

$D_{UBCBookStore}$  
The XML Database for UBCBookstore, 4

$\Delta$  
a DTD definition, 1

$\Phi$  
a function that returns child DTD elements of a DTD element, 11

$\Pi$  
a function that returns DTD attributes of a DTD element, 11

$\Sigma$  
A set of Constraints, 2

$\equiv$  
operator that states two XML databases contain the same collection of data, 27

$\mu$  
function to transform XML to relational, 25

$\mu^{-1}$  
function to reverse transform relational to XML, 25

$\tau$  
a DTD element, 11

$A$  
a set of DTD attributes in a DTD, 11

acyclic  
, 12

att  
a function that returns a attribute of an element, 16

$C$  
A Set of Correspondences, 7

choice-free  
, 11
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>an XML database, 16</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition, 1</td>
</tr>
<tr>
<td>DTD group</td>
<td>, 15</td>
</tr>
<tr>
<td>duplicate-free</td>
<td>, 13</td>
</tr>
<tr>
<td>E</td>
<td>a set of DTD elements in a DTD, 11</td>
</tr>
<tr>
<td>FD</td>
<td>Functional Dependency, 1</td>
</tr>
<tr>
<td>IC</td>
<td>Integrity Constraint, 1</td>
</tr>
<tr>
<td>IND</td>
<td>Inclusion Dependency, 1</td>
</tr>
<tr>
<td>intersection</td>
<td>, 23</td>
</tr>
<tr>
<td>lab</td>
<td>a function that returns the DTD element of an element, 16</td>
</tr>
<tr>
<td>M</td>
<td>a set of HepToX mapping rules, 19</td>
</tr>
<tr>
<td>proper circular</td>
<td>, 22</td>
</tr>
<tr>
<td>r</td>
<td>the root DTD element of a DTD, 11</td>
</tr>
<tr>
<td>reduced</td>
<td>, 24</td>
</tr>
<tr>
<td>root</td>
<td>the root element of an XML database, 16</td>
</tr>
<tr>
<td>split-freeness</td>
<td>, 23</td>
</tr>
<tr>
<td>typed relational IND</td>
<td>, 20</td>
</tr>
<tr>
<td>typed XML IND</td>
<td>, 21</td>
</tr>
<tr>
<td>unorderness</td>
<td>, 13</td>
</tr>
<tr>
<td>V</td>
<td>a finite set of elements in an XML database, 16</td>
</tr>
</tbody>
</table>

x
XML

Extensible Markup Language, I
Chapter 1

Introduction

Extensible Markup Language (XML) has been the dominating data representation format on the Internet. It is self-describing and semi-structured; thus giving applications great flexibility to implement and interpret data. These properties make it ideal for data exchange, and it has been the dominating format for data exchange.

However, we believe exchanging of data on the Internet does not utilize the full potential of XML. One way to further expand the power of data exchange is to examine the semantics of data in order to create better quality mappings which lead to better data exchange.

As with most relational databases, there are integrity constraints (ICs) such as functional dependencies (FDs) and inclusion dependencies (INDs) that give the data more semantics than just tuples of values. There are already studies of ICs in XML such as [6, 10]. Therefore, the next step is to understand how these semantics can benefit data exchange.

Although there are many different types of ICs, FDs and INDs are the most used ones. The two assist in storage, update, querying, and transformation of data. As a result, they are very heavily integrated into everyday applications nowadays. If applications are running on data that violates these constraints, they may behave erratically and return unsound and unexpected results.

We demonstrate some of the challenges we face using the example in Figure 1.1. In Figure 1.1, there are two Document Type Definitions (DTDs), $\Delta_{\text{UBCBookstore}}$ and $\Delta_{\text{UBCLibrary}}$. A DTD is a schema which defines the syntax of XML databases.
In our example, the two DTDs are expressed graphically as DTD graphs (DTD trees in our example). Each graph contains nodes as DTD elements, and, for simplicity, all leaf nodes (nodes without children) are DTD attributes.

We have established a few correspondences, shown in arrows, from the attributes in the source UBCBookstore to the attributes in the target UBCLibrary. These simple 1-1 correspondences.

We can also define ICs on a DTD. The FDs and INDs for UBCBookstore are listed in Table 1.1, and the ones for UBCLibrary are in Table 1.2.

$\Delta_{UBCBookstore}$ keeps books under different subjects. BookFD#3-9 in $\Sigma_{UBCBookstore}$ state that there is an isbn that implies all the other attributes within every book. In

<table>
<thead>
<tr>
<th>Table 1.1: $\Sigma_{UBCBookstore}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BookFD#1: UBCBookstore.person.name $\rightarrow$ DOB</td>
</tr>
<tr>
<td>BookFD#2: UBCBookstore.subject.book.author.name $\rightarrow$ DOB</td>
</tr>
<tr>
<td>BookFD#3: UBCBookstore.subject.book.isbn $\rightarrow$ title</td>
</tr>
<tr>
<td>BookFD#4: UBCBookstore.subject.book.isbn $\rightarrow$ year</td>
</tr>
<tr>
<td>BookFD#5: UBCBookstore.subject.book.isbn $\rightarrow$ price</td>
</tr>
<tr>
<td>BookFD#6: UBCBookstore.subject.book.isbn $\rightarrow$ #pages</td>
</tr>
<tr>
<td>BookFD#7: UBCBookstore.subject.book.isbn $\rightarrow$ pub</td>
</tr>
<tr>
<td>BookFD#8: UBCBookstore.subject.book.isbn $\rightarrow$ author.name</td>
</tr>
<tr>
<td>BookFD#9: UBCBookstore.subject.book.isbn $\rightarrow$ author.DOB</td>
</tr>
</tbody>
</table>

| BookIND#1: UBCBookstore.subject.book.author[name, DOB] $\subseteq$ UBCBookstore.person[name, DOB] |
addition, there is a main author listed under every book, and by BookIND#1, these authors are also listed independently as persons under the root node UBCBookstore.

Meanwhile, the $\Delta_{UBCLibrary}$ has a slightly flatter structure. Each Publication is listed under the root node UBCLibrary, and all its information is listed as attributes under its subtree. In a typical library, subject, author.name, and year of a publication together determines the call number of a publication, which is a unique number that libraries use to identify publications. Similar to $\Delta_{UBCBookstore}$ above, there is a main author listed under each publication, and the same author is also listed as person under root. Similarly, each publisher listed as pub under publications is also listed under publisher under root.

The two heterogeneous schemas have different hierarchies. For example, books in UBCBookstore are grouped by subjects, while subjects in UBCLibrary are listed under publications. There are also attributes that exist in one schema but not the other (e.g. publisher.addr in UBCLibrary).

The two schemas also represent heterogeneous content. UBCBookstore keeps a record of books, but UBCLibrary records publications which subsume books. In addition, the retailer background of UBCBookstore is shown by the presence of

| LibFD1: | UBCLibrary.person: name → DOB |
| LibFD2: | UBCLibrary.person: name → addr |
| LibFD3: | UBCLibrary.publication: call#.author.name → call#.author.DOB |
| LibFD4: | UBCLibrary.publication: |
| LibFD5: | UBCLibrary.publication: |
| LibFD6: | UBCLibrary.publication: |
| LibFD7: | UBCLibrary.publisher: name → addr |
| LibIND1: | UBCLibrary.publication.call#.author[name, DOB] ⊆ UBCLibrary.person[name, DOB] |
| LibIND2: | UBCLibrary.publication[pub] ⊆ UBCLibrary.publisher[name] |
price which is absent in UBCLibrary for obvious reasons.

Heterogeneity of the two schemas is not limited to the representation of the data. The two sets of constraints also show variations.

In UBCBookstore, book.title and book.#pages are uniquely identified by book.isbn as specified by the FDs BookFD#3 and BookFD#6. However, UBCLibrary uses publication.subject, publication.author.name, and publication.year to imply the values of publication.title and publication.#pages as specified in LibFD#4 and LibFD#5. Thus, problems arise if two books having the same subject, author name, and year but different #pages or titles in UBCBookstore are translated to instances of publication in UBCLibrary. The two translated publication instances will violate the FDs LibFD#4 and LibFD#5.

Suppose that UBCBookstore has stacks of unsold books, and decides to donate them to UBCLibrary. Along with the books, UBCBookstore also gives UBCLibrary an XML database $D_{UBCBookstore}$ containing information about the books it is donating. $D_{UBCBookstore}$ conforms to DTD UBCBookstore and its constraints. Upon receiving $D_{UBCBookstore}$, librarians at UBCLibrary want to integrate the data into its database which conforms to DTD UBCLibrary and its constraints.

The librarians understand that the UBCBookstore data does not conform to its DTD, and it will be tedious work and error-prone to translate the data manually. In addition, there will probably be frequent donations from the UBCBookstore in the future. Therefore, an automated translation framework from data in UBCBookstore to UBCLibrary will be appreciated.

The librarians can create high-level correspondences from the UBCBookstore DTD to the UBCLibrary DTD. Moreover, using methods in [5] and [14], they can generate mapping rules from these correspondences. The mapping rules translate data in UBCBookstore to data that conforms to UBCLibrary DTD but not necessary its constraints.

The XML instance in Figure 1.2 conforms to the DTD $\Delta_{UBCBookstore}$ and its constraints. With the mapping rules in Figure 1.4 which are generated by algorithms in [5], the librarians can translate data in Figure 1.2 to the data in Figure 1.3.

The translated document in Figure 1.3 conforms to the DTD UBCLibrary, but it does not conform to the UBCLibrary constraints. It is obvious that pub of each
instance of publication is not listed under publisher.name as LibIND#2 of UBCLibrary requires. Moreover, the two publications have the same subject, author.name, and year, but different title and #pages. This violates the FDs LibFD#4 and LibFD#5 of UBCLibrary.

Applications in the library use subject, author.name, and year to identify publication.title, but given data in Figure 1.3 these applications will no longer behave correctly.

Clearly, something needs to be done to resolve this problem. Fixing constraint violation after data is translated using existing constraint violation repair technique such as [3, 4] can be undesirable because UBCLibrary may not want to alter data.

Correspondences are usually created by data administrators who know both the source and target schemas relatively well. If we can detect constraint violations at the time of correspondence establishment, and provide suggestions on how to repair the correspondences, such administrators are given more information and arsenals to create better quality correspondences.

This thesis aims to detect such violations and suggests ways to repair them.
1. \text{UBCLibrary} \rightarrow \text{f1}($\text{UBCBookstore}$)[
    person \rightarrow \text{f2}($\text{name}, \text{DOB}$)[
        \text{name} \rightarrow \text{name},
        \text{DOB} \rightarrow \text{DOB}]
]

\text{UBCBookstore} \rightarrow \text{UBCBookstore}[*
    \text{name} \rightarrow \text{name},
    \text{DOB} \rightarrow \text{DOB}]*

2. \text{UBCLibrary} \rightarrow \text{f1}($\text{UBCBookstore}$)[
    \text{publication} \rightarrow \text{f3}($\text{sname}, \text{aname}, \text{DOB}, \text{title}, \text{year}, \text{pages}, \text{pub}$)[
        \text{call}.\text{subject} \rightarrow \text{sname},
        \text{call}.\text{author.name} \rightarrow \text{aname},
        \text{call}.\text{author.DOB} \rightarrow \text{DOB},
        \text{call}.\text{year} \rightarrow \text{year},
        \text{title} \rightarrow \text{title},
        \text{#pages} \rightarrow \text{pages}]]

\text{UBCBookstore} \rightarrow \text{UBCBookstore}[*
    \text{subject} \rightarrow \text{sname}[*
        \text{name} \rightarrow \text{sname},
        \text{book} \rightarrow \text{book}[*
            \text{title} \rightarrow \text{title},
            \text{#pages} \rightarrow \text{pages},
            \text{author.name} \rightarrow \text{aname},
            \text{author.DOB} \rightarrow \text{DOB},
            \text{pub} \rightarrow \text{pub}]]*

\text{Figure 1.4: Mapping Rules M for UBCBookstore} \rightarrow \text{UBCLibrary Example}
Meanwhile, every algorithm has to be efficient in order for our framework to be practical in the real world.

**Problem Statement**

Given a set of constraints \( \Sigma_S \) over source DTD \( \Delta_S \), a set of constraints \( \Sigma_T \) over target DTD \( \Delta_T \), and a set of user-defined correspondences \( C \), determine if \( \forall \) XML databases \( D_S \models \Sigma_S \) over \( \Delta_S \), after translation via \( C \), the translated instances satisfy \( \Sigma_T \).

In addition, if violations exist, how do we fix them?

**Related Work and Challenges**

In order to implement an XML constraint translation system, an essential building block is solving constraint implication efficiently. However, as shown in [13] and [8], implication of relational FD and IND considered together is undecidable. Therefore, it is challenging to find an efficient XML constraint implication system.

[10] shows interaction of XML constraints with DTDs, it finds keys and foreign keys consistency with DTDs undecidable. Thus taking DTD interactions out of the equation is essential to a tractable reasoning with XML constraints. We simplify DTDs in this thesis aiming to avoid such interactions.

[9] propagates XML keys to FDs on a predefined relational database via a transformation. It shows XML key and foreign key propagation is undecidable; therefore, it only focuses on XML keys. It further emphasizes the difficulties in the problem of XML constraint propagation.

[14] generates join satisfying mapping rules from user defined correspondences if the two schemas have identical representation of entities. It considers join conditions on both the source and target schemas such that the mapping rules generated satisfy target join constraints. However, both keys and joins in the target can be violated if suspicious correspondences are present. They can be mistakes of the user who defines the correspondences, or they may be unavoidable due to different representations of entities in of the two schemas.

The absence of correspondences creates ‘holes’ in the target. These missing values or NULL values further complicate the propagation problem. In addition,
inability to propagate does not necessarily imply constraint violations, and these missing or NULL values play a big part in determining violations.

Finally, once violations are detected, there are different ways to fix them. Each has its advantages and disadvantages; the combinations of these repairs give users choices to fix violations under different circumstances. Existing constraint violation repair techniques such as [3],[4] treat the translated data as one inconsistent database, and either delete, add modify tuples to make data adhere to the target constraints. In trying to make data conform to the target constraints, modification of translated data without knowing anything about the originating data may sacrifice the correctness. In contrast, our solution of correcting the correspondences offers a different approach to preserve the coherence of data with the intelligence of the originating data structure together with its semantics.

Contributions

1. There currently does not exist an implication testing system for XML FDs and INDs considered together; therefore, we reduce XML constraint implication to relational constraint implication so that we can employ tools in well studied relational constraint reasoning.

2. We develop a polynomial algorithm to test XML constraint implication.

3. We develop a polynomial algorithm to test XML constraint propagation.

4. We develop a polynomial algorithm to detect XML constraint violation.

5. We suggest ways to repair mappings should a violation occur.

Organization

We present the rest of the document as follows. Chapter 2 lays the foundation for the later chapters by specifying definitions. Chapter 3 describes the reduction of XML constraint implication to relation constraint implication, plus how to make the implication problem tractable. We develop the constraint propagation algorithm in Chapter 4, and show that it runs in polynomial time. Chapter 5 builds constraint violation detection algorithm on top of the propagation algorithm.
Chapter 6 suggests ways of repairing mappings such that constraints will not be violated. Chapter 7 concludes our work and offers a glimpse of how this work can be extended.
Chapter 2

Definitions

2.1 DTD definition

Majority of this definition is adopted from [2].

Definition 2.1.1 A DTD (Document Type Definition) is defined to be: \( \Delta = (E, A, \Phi, \Pi, r) \), where:

- \( E \) is a finite set of element types.
- \( A \) is a finite set of (non-set) attributes.
- \( \Phi \) is a mapping from \( E \) to element type definitions: Given \( \tau \in E \), \( \Phi(\tau) = S \), where \( S \) is a string type, or \( \Phi(\tau) \) is a regular expression \( \alpha \) defined as follows:

\[
\alpha ::= \varepsilon | \tau' | \alpha | \alpha, \alpha | \alpha? | \alpha* | \alpha^+ 
\]

where \( \varepsilon \) is the empty sequence, \( \tau' \in E \), \( '|' \) represents union, \( ',' \) represents concatenation, \( '?' \) represents zero or one occurrence of \( \alpha \), \( '*' \) represents zero or more occurrences of \( \alpha \) (Kleene closure), and \( '^+' \) represents one or more occurrences of \( \alpha \).

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

For example, we can present \( \Delta_{UBCBookstore} \) in Figure 1.1 in the form \((E,A,\Phi,\Pi,r)\), where:

- \( E = \{UBCBookstore, subject, name, book, author, person\} \)
- \( A = \{name, isbn, title, year, price, #pages, DOB, pub\} \)
- \( \Phi = \{UBCBookstore \rightarrow (subject*, person*), subject \rightarrow (book*), book \rightarrow (author)\} \)
- \( \Pi = \{subject \rightarrow (name), book \rightarrow (isbn, title, year, price, #pages, pub), author \rightarrow (name, DOB), person \rightarrow (name, DOB)\} \)
- \( r = UBCBookstore \)

### 2.2 Simple DTD definition

In order for the algorithms in the rest of this thesis to be efficient, we simplify our DTD definition.

As shown in [10], a DTD interacts with XML constraints to produce inconsistent specifications. That is, there does not exist an XML database that satisfies an inconsistent set of DTD and XML constraints. On the contrary, a consistent set of DTD and XML constraints is one that has at least one XML database that satisfies it. To make matters worse, [12] shows that a DTD by itself may be inconsistent too.

We wish to guarantee no interaction between DTDs and XML constraints. However, a full-blown study of this topic is beyond the scope of our research. Therefore, we specify choice-free and acyclic DTDs in the following section; they together guarantee DTDs are by themselves consistent as proven in [12].

In addition, [10] shows that cardinality constraints of DTDs interact with XML keys and foreign keys. Therefore, simplifying cardinality constraints should eliminate or reduce interactions between DTDs and sets of XML FDs and INDs. We achieve this by imposing the properties duplicate-free and unorderness on DTDs.
2.2.1 Choice-free

[12] proves choice-free is one of the properties required for a consistent DTD $\Delta = (E, A, \Phi, \Pi, r)$. We redefine this property under our context.

Choice-free is the elimination of choices (options) of child DTD elements. To eliminate the 'choice' of child nodes, we redefine the regular expression allowed for $\Phi(\tau)$ in our DTD definition $\Delta = (E, A, \Phi, \Pi, r)$, where $\tau \in E$. The original regular expression $\alpha$ of $\Phi(\tau)$ is:

$$\alpha ::= e | \tau' | \alpha | \alpha | \alpha? | \alpha^* | \alpha^+ S \tag{2.1}$$

... The '|' (union) operator gives $\Phi(\tau)$ the 'choice' property. Eliminating '|' from $\alpha$:

$$\alpha ::= e | \tau' | \alpha | \alpha | \alpha? | \alpha^* | \alpha^+ \tag{2.2}$$

makes our new DTD definition satisfy the choice-free property.

2.2.2 Acyclic

[12] shows that a DTD without cycles is another requirement for a consistent DTD. A DTD $\Delta = (E, A, \Phi, \Pi, r)$ is cyclic if $\Delta$ contains a DTD Element $\tau \in E$ such that $\tau$ appears as a descendant of itself. The DTD graph of $\Delta$ would contain a cycle with $\tau$ being one of the nodes of the cycle.

Another benefit of having acyclic DTDs is the elimination of infinite paths. With a DTD containing cycles, it is possible to have a path of elements starting from the root that loops a cycle forever. The elimination of these infinite paths is important because we will later use fully specified paths to define our constraints. Having cycles in DTDs may allow constraints with infinite depth, and infinite number of constraints. Both are very undesirable properties, and will make reasoning with such constraints intractable.

To incorporate the acyclic property into our DTD language definition, we need to add an extra statement to the definition of the regular expression language for $\Phi(\tau)$ in (2.2), where $\tau \in E$. The new definition is:
\[ \alpha ::= \epsilon \mid \tau' \mid \alpha, \alpha \mid \alpha^* \mid \alpha^+ \quad (2.3) \]

where \( \tau' \not\in \text{ancestor}(\tau) \), and \( \tau' \neq \tau \)

2.2.3 Duplicate-free

We define a new property \textit{duplicate-free}, that aims to simplify cardinality constraints that can be implied by DTDs.

In a DTD definition \( \Delta = (E, A, \Phi, \Pi, r) \), \( \Phi(\tau) \) allows a child \( \tau' \in E \) to appear multiple times; therefore, \( \tau' \) can have minimum and maximum occurrence constraints of any integers. For example, if \( \Phi(\tau) = \tau', \tau', \tau', \tau', \tau' \), then \( \tau' \) has to appear at least three times under \( \tau \), and can appear a maximum of six times.

In [10], it shows these cardinality constraints can interact with XML constraints to produce inconsistent specifications; therefore, we introduce \textit{duplicate-free} to simplify cardinality constraints. The idea is simple; we do not allow a child to appear multiple times in a \( \Phi(\tau) \) regular expression. Together with the \textit{unorderness} property in Section 2.2.4, we will define a new regular expression language for \( \Phi \) in our DTD definition.

The \textit{duplicate-free} property only allows child DTD elements to have minimum occurrences of either 0 or 1, and maximum occurrences of either 1 or \( \infty \). It greatly simplifies the complexity of constraint reasoning.

2.2.4 Unorderness

Subelements in XML are \textit{ordered}. For example:

for an root element \( \text{Root}_1 \), and \( \Phi(\text{Root}_1) = BC \),
\[
T_1 = \langle A >\langle B/ >\langle C/ >\langle /A >
\]

for another root element \( \text{Root}_2 \), and \( \Phi(\text{Root}_2) = CB \),
\[
T_2 = \langle A >\langle C/ >\langle B/ >\langle /A >
\]

\( T_1 \) and \( T_2 \) differ in their order of elements; thus \( T_1 \neq T_2 \). However, current constraints view data as collections of data in which order does not matter. We aim to employ current constraint reasoning techniques; hence, for the rest of this thesis, we will ignore the order of elements. Thus, we achieve \textit{unorderness}. 


Together with the duplicate-free property, we redefine the regular expression language for \( \Phi(\tau) \) in our DTD definition \( \Delta = (E, A, \Phi, \Pi, r) \), where \( \tau \in E \). The regular expression \( \alpha \) of \( \Phi(\tau) \) in (2.3) is:

\[
\alpha ::= e | \tau' | \alpha, \alpha | \alpha? | \alpha* | \alpha^+
\]

(2.4)

where \( \tau' \notin ancestor(\tau) \), and \( \tau' \neq \tau \)

Now that we can ignore order, and have to honour the duplicate-free property, we can simplify the regular expression language to a list of unique DTD elements, each is followed by a quantifier:

\[
\alpha ::= e | \tau_1[1 | ?| ^*| ^+] | \ldots | \tau_n[1 | ?| ^*| ^+]
\]

(2.5)

where \( \forall i, j \in [1 \ldots n] \), and \( i \neq j \), \( \tau_i \neq \tau_j \neq \tau \), and \( \tau_i, \tau_j \notin ancestor(\tau) \).

The quantifier ‘1’ behaves exactly as if there were no quantifiers. It states that the child DTD element has both minimum and maximum occurrence of 1.

In Section 3.1.2, we define XML database equivalence that further elaborates on how we ignore order.

### 2.3 DTD Group

We divide the set of DTD elements into groups to give the XML constraints a platform to apply their logic. This is the same as ‘group’ in [5], and analogous to logical relations in [14].

We are going to define single path in the next section, then use it as a building block to define DTD group in the section that follows.

#### 2.3.1 Single Path

Let \( \Delta = \{E, A, \Phi, \Pi, r\} \) be a DTD defined in Definition 2.1.1. An edge is a relation from a parent DTD element \( \tau \in E \) to a child DTD element defined in \( \Phi(\tau) \), or it is a relation from \( \tau \) to an attribute defined in \( \Pi(\tau) \). Such edges are single if it is from an element to an attribute (because we don’t allow set attributes), or if it is between
two elements with the cardinality constraint of ‘?’ or ‘1’. On the contrary, edges between elements with the cardinality constraint of ‘*’ or ‘+’ are multiple edges.

Edges link together to form paths, and a single path (SP) is a path that consists of single edges only. In addition, a single path always ends in a DTD leaf text node or a DTD attribute. SP is expressed as a fully specified relative XPath between an ancestor and a descendant.

2.3.2 DTD Group

Definition 2.3.1 A DTD group is a group of DTD elements and attributes that are all connected by single paths.

A DTD group guarantees one-to-one relations within a branch of elements and attributes in an XML database that have their DTD elements and DTD attributes within the same group. Such a branch of elements and attributes is analogous to a relational tuple.

![Diagram of UBCBookstore DTD Group](image)

Figure 2.1: $\Delta_{\text{UBCBookstore}}$

For the rest of this thesis, we will use $G$ to denote a DTD Group. And we label each group with the path name of the highest node $\tau$, where $\Phi(parent(\tau))$ contains
either $\tau^*$ or $\tau^+$. We also conveniently call $\tau$ the DTD Group Node of $G$.

Figure 2.1 shows the 4 DTD groups in $\Delta_{\text{UBCBookstore}}$. For example, subject and book do not belong to the same DTD group because the edge between subject and book is not a single path; it has one of the quantifiers that allow multiple occurrences of books in each subject. On the contrary, there is a single path between book and DOB; therefore, they are in the same DTD group.

### 2.4 XML Database

We adopt the following definition of XML databases from [2].

An XML database $D$ conforming to a DTD $\Delta = (E, A, \Phi, \Pi, r)$ is defined to be a tree $(V, \text{lab}, \text{ele}, \text{att}, \text{root})$, where

- $V$ is a finite set of vertices (nodes).
- $\text{lab} : V \rightarrow E$.
- $\text{ele} : V \rightarrow \text{String} \cup V^*$.  
- $\text{att}$ is a partial function $V \times A \rightarrow \text{String}$. For each $v \in V$, the set $\{l \in A | \text{att}(v, l) \text{ is defined}\}$ is required to be finite.
- $\text{root} \in V$ is called the root of $d$.

#### 2.4.1 Simplifying text nodes to attributes

Since all the leaf nodes are either DTD elements containing text or DTD attributes, for simplicity without loss of generality, we assume all leaf nodes are DTD attributes for the rest of this document.

#### 2.4.2 Notations

For any element $x \in V$ in $D$ labelled by $\tau \in E$, $\text{lab}(x) = \tau$, and any path $P$ originating from $\tau$, $x.P$ returns all elements that are descendents of $x$ and are reachable from $x$ via the path $P$ if $P$ ends in a DTD element, it returns string values that are values of attributes reachable from $x$ via the path $P$ if $P$ ends in a DTD attribute.

For example, let's call the only subject element in Figure 1.2 $x$. Then, $x.book.price$ returns "120", "70".
2.5 Missing Values and NULL Values

For the rest of this thesis we treat missing values as if they are NULL values. NULL values are known to be incomparable; that is they do not equate to any values, not even to other NULL values.

NULL values are going to play a big role in how we treat FDs and INDs.

2.6 XML constraint definitions

2.6.1 Functional dependency

An XML functional dependency, $\sigma_{FD}$ has the form:

$$\tau : SP_1, \ldots, SP_n \rightarrow SP_u$$

where $\tau \in E$, and $\tau$ is a DTD group node. In addition, $SP_1, \ldots, SP_n, SP_u$ are single paths originating from $\tau$ and ending in attributes.

An XML database $D=(V, \text{lab}, \text{ele}, \text{att}, \text{root})$ satisfies $\sigma_{FD}$, denoted:

$$D \models \sigma_{FD} \text{ iff }$$

$$\forall x, y \in V, \text{lab}(x) = \text{lab}(y) = \tau,$$

$$\bigwedge_{i \in 1, \ldots, n} (x.SP_i = y.SP_i) \rightarrow x.SP_u = y.SP_u$$

Note that we cannot equate NULL values to any other values; thus NULL existence on the L.H.S. guarantees FD satisfaction. In [9], FDs are defined in participation of them turning into keys. Such keys are then used to normalize relational schemas. Since keys must be able to identify tuples, incomplete keys are unacceptable. Therefore, [9] requires an instance to have NULL value for the R.H.S. of an FD if there exists NULL value among the values of the L.H.S. attributes.

However, unlike [9], we want to reason only with the most fundamental form of ICs only, FDs and INDs. Therefore, we can concentrate on these two types of ICs only when we consider NULL values. FDs and INDs reason with leaf values only, but not node identities as in keys. Consequently, an instance with NULL values on the L.H.S. of an FD guarantees satisfaction. On the other hand, if the
valuation of the R.H.S. in an instance is NULL, we treat it as a unique value, and only require its valuation of the L.H.S. to differ from all other instances.

### 2.6.2 Inclusion dependency

An XML inclusion dependency, $\sigma_{IND}$ has the form:

$$\tau_1[SP_{1,1},\ldots,SP_{1,n}] \subseteq \tau_2[SP_{2,1},\ldots,SP_{2,n}]$$

where $\tau_1, \tau_2 \in \mathcal{E}$, and $\tau_1, \tau_2$ are DTD group nodes. In addition, $SP_{1,1},\ldots,SP_{1,n}$ and $SP_{2,1},\ldots,SP_{2,n}$ are single paths in $\tau_1$ and $\tau_2$ respectively.

An XML database $D=(V,\text{lab},\text{ele},\text{att},\text{root})$ satisfies $\sigma_{IND}$, denoted:

$$D \models \sigma_{IND}$$

iff

$$\forall x \in V, \text{lab}(x) = \tau_1, \exists y \in V, \text{lab}(y) = \tau_2, \text{s.t.}$$

$$\bigwedge_{i \in \{1,\ldots,n\}} x.SP_{1,i} = y.SP_{2,i}$$

Note that missing values or NULL values are not allowed on either side of an IND because such values are incomparable; thus would violate the IND.

### 2.7 Schema Mapping from HepToX

In order to translate data from a source DTD $\Delta_S$ to a target DTD $\Delta_T$, we need a mapping language to define how to translate. Both HepToX [5] and Clio [14] can derive schema mappings from simple correspondences drawn as arrows from attributes of source schema to those in the target.

From correspondences, [14] generates inter-schema referential constraints in a form similar to SQL statements. Although, such constraints define how data translate, they are too technical and thus are difficult to analyse.

On the other hand, users in [5] create correspondences between source and target leaf nodes. These correspondences are used to generate free-log mapping rules that define how data from the source can be translated to the target. These mapping rules are in the form of tree expressions, as such they are more suitable for analysis for the rest of this document.
However, our mapping model differs slightly from [5] and [14] in that we do not employ the ID/IDREF function of DTD. In other words, we do not have key and foreign key constraints in our model. Instead we deal with the more fundamental constraints: FD and IND. Thus there are no joins among DTD groups, and our mapping rules only translate instances along a DTD tree path (attributes along a path from the root to the group node in question).

We give a brief outline here on how simple correspondences in our running example in Figure 1.1 become a set of mapping rules M in Figure 1.4 following algorithms in [5].

The target DTD UBCLibrary is broken into 3 DTD groups, the groups can be seen as logical relations in which attributes (plus leaf text nodes) together describe an entity. The 3 DTD groups are publication, person, and publisher, and they each become the target of a mapping rule except for publisher because none of its attributes are mapped.

On the other hand, the source schema is broken into 3 groups: subject, book, and person. In each mapping rule, if the target group has any correspondences that originates from a particular source group, then the source group is included in the rule body of the rule along with its ancestors.

Each mapped attribute in the rule body is bound to a unique variable, and its correspondence in the target is assigned that variable. This has the same function as in SQL where tables are bound to some variables in the FROM clause, then some of the variables appear in the SELECT clause to create view instances by retrieving data from the tables in which the variables are bound.

In mapping rule #1, every person's name and DOB from the source is put into name and DOB in person in the target. The rule is very straightforward. However, notice the target root is assigned a Skolem function $fl($)UBCBookstore$), and $UBCBookstore$ is the binding variable of the root element in the source; thus translated instances will always be stitched together and have only one root.

The function of our Skolem function is to assign non-leaf nodes an identity. Thus two non-leaf nodes with the same identity (same Skolem function with the same arguments), will be merged into one node.

Mapping rule #2 takes book instances from the source, and put them into publication at the target. The rule also acquires the name of subject which is in the
2.8 Relational Constraint Definitions

In this section, we quickly define the relational constraints: FD and IND; [1] provides a more comprehensive explanation of FD and IND. We will be using relational FD and IND implication to solve the XML FD and IND implication problem; therefore, we will introduce them in the following sections.

In addition, in order to solve our XML constraint implication problem efficiently, we use typed relational IND which can be solved in polynomial time. We will also define typed relational IND in this section.

2.8.1 Relational Functional Dependency

Let $R$ be a relational schema, and $A_1, \ldots, A_n, A_u$ be $n+1$ attributes in $R$. A relational FD $\sigma_R$ has the form:

$$R.A_1 \ldots A_n \rightarrow A_u$$

A relational database $D_R$ conforming to $R$ satisfies $\sigma_R$ if $\forall$ tuples $t_1, t_2$ in table $R$ of $D_R$,

$$\bigwedge_{i \in 1, \ldots, n} (t_1.A_i = t_2.A_i) \rightarrow t_1.A_u = t_2.A_u$$

2.8.2 Relational Inclusion Dependency

Let $R_1, R_2$ be two relation schemas in a database schema $R$, $A_1, \ldots, A_n$ be $n$ attributes in $R_1$, and $B_1, \ldots, B_n$ be $n$ attributes in $R_2$. A relational IND $\sigma_R$ has the form:

$$R_1.A_1 \ldots A_n \subseteq R_2.B_1 \ldots B_n$$

A relational database $D_R$ conforming to $R$ satisfies $\sigma_R$ if $\forall$ tuple $t_1$ in table $R_1$ of $D_R$ there exists a tuple $t_2$ in table $R_2$ of $D_R$ such that:

$$\bigwedge_{i \in 1, \ldots, n} (t_1.A_i = t_2.B_i)$$
2.8.3 Typed Relational IND

A more comprehensive study of Typed Relational IND can be found in [7]. We give a quick definition and description of Typed Relational IND here.

Definition 2.8.1 Let $R_1, R_2$ be two relation schemas in a database schema $R, A_1, \ldots, A_n$ be $n$ attributes in $R_1$, and $B_1, \ldots, B_n$ be $n$ attributes in $R_2$. A typed relational IND $\sigma_R$ has the same form as a typical relation IND:

$$R_1.A_1 \ldots A_n \subseteq R_2.B_1 \ldots B_n$$

The “typed” property requires $\forall i \in [1, \ldots, n] \ A_i = B_i$.

Moreover, any two DTD attributes $A'_1, A'_2$ within each relation schema $R' \in R$ must not be of the same attribute type.

A relational database $D_R$ conforming to $R$ satisfies $\sigma_R$ just as it would for a typical relational IND.

In more details, there exists a pool of unique attributes $U$ from which each relation schema $R_1, R_2$ gets a subset of attribute. As a result, attributes of a relation schema are unique within the schema. Moreover, if $A_i=B_i$, where $A_i$ is an attribute of $R_1$ and $B_i$ is an attribute of $R_2$, then $A_i$ and $B_i$ are the same attribute in $U$.

2.9 Typed XML Inclusion Dependency

We model each DTD group as an entity, same as each relation schema represents an entity. In order to integrate the typed property, each DTD attribute appears at most once in a DTD group. Moreover, pairs of corresponding attributes on the L.H.S. and R.H.S. of an XML IND must be of the same DTD attribute type.

We now give a formal definition on how to integrate the typed property into a DTD $\Delta$.

Definition 2.9.1 Let $\Delta = (E, \Lambda, \Phi, \Pi, r)$, $\forall \tau_i, \tau_j \in E$, if $\tau_i, \tau_j$ belong to the same DTD group $G$, then $\Pi(\tau_i) \cap \Pi(\tau_j) = \emptyset$.

Definition 2.9.2 Let $\sigma$ be an XML IND in the form:

$$\tau_1[SP_{1,1}, \ldots, SP_{1,n}] \subseteq \tau_2[SP_{2,1}, \ldots, SP_{2,n}]$$
where $\tau_1, \tau_2 \in E$, $\tau_1, \tau_2$ are DTD group nodes, $SP_{1,1}, \ldots, SP_{1,n}$ and $SP_{2,1}, \ldots, SP_{2,n}$ are single paths in $\tau_1$ and $\tau_2$ respectively. $\sigma$ is typed iff

$$\bigwedge_{i \in 1, \ldots, n} (\tau_1 \cdot SP_{1,i} = \tau_2 \cdot SP_{2,i})$$

### 2.10 Guaranteeing No Interaction of relational FD and IND Implication

In order to find an efficient XML constraint implication system, we first look at the implication of the simpler relational constraints.

As mentioned earlier, implication of FDs and INDs together is undecidable [8] [13]. The complication of the implication comes from the interaction between the two kinds of constraints. Thus if we can avoid the interaction of FDs and INDs, it will be a big step towards making the implication problem tractable.

[11] proves it is possible to prevent such interaction albeit imposing some restrictions on the FDs and INDs. We quote each condition as defined in [11] in the following sections, and give each a short description. Finally, in Section 2.10.5, we present the theorem from [11] that guarantees non-interaction.

#### 2.10.1 Proper Circular

**Definition 2.10.1** A set of INDs $I$ over relational schemas $R$ is circular if either there exists a nontrivial IND $R[X] \subseteq R[Y] \in I$, where $X, Y$ are arrays of attributes in $R$, or there exists $m$ distinct relation schemas, $R_1, R_2, \ldots, R_m \in R$, with $m > 1$, such that $I$ contains the INDs: $R_1[X_1] \subseteq R_2[Y_2], R_2[X_2] \subseteq R_3[Y_3], \ldots, R_m[X_m] \subseteq R_1[Y_1]$, where $X_i, Y_i$ are sets of attributes in $R_i$. A set of INDs $I$ is noncircular if it is not circular.

**Definition 2.10.2** $I$ is proper circular if it is either noncircular or whenever there exit $m$ distinct relation schemas, $R_1, R_2, \ldots, R_m \in R$, with $m > 1$, such that $I$ contains the INDs: $R_1[X_1] \subseteq R_2[Y_2], R_2[X_2] \subseteq R_3[Y_3], \ldots, R_m[X_m] \subseteq R_1[Y_1]$, then for all $i \in \{1, 2, \ldots, m\}$ we have $X_i = Y_i$; such a sequence of INDs is called a proper cycle.
Intuitively, I is circular if a subset of its INDs forms a circle in a graph \((V,E)\), where each relation \(R_i\) is a vertex \(V_i\) in \(V\) and each IND \(R_i[X_i] \subseteq R_j[Y_j]\) is an edge from \(V_i\) to \(V_j\). I satisfies proper circular if I is not circular, or if a cycle exists, at each vertex, the sequences of attributes \(X_i, Y_i\) connecting the two edges are two identical sequences of attributes.

### 2.10.2 Split-freeness

Two nontrivial FDs of the forms \(R:XB \rightarrow A\) and \(R:YA \rightarrow B\), where \(R\) is a relation schema, \(X,Y\) are sets of attributes of \(R\), and \(A,B\) are two attributes of \(R\), are said to be cyclic.

A closure of a set of FDs \(F\), denoted \(F^+\), is a set of FDs that contains every FD \(\sigma\) where \(F \models \sigma\).

**Definition 2.10.3** A set of FDs \(F\) over \(R\) satisfies the split-freeness property if \(\forall F_j \in F\), whenever there exist cyclic FDs, \(R_j:XB \rightarrow A, YA \rightarrow B \in F_j^+\), then either \(R_j:Y \rightarrow B \in F_j^+\) or \(R_j:(X \cap Y)A \rightarrow B \in F_j^+\).

Roughly speaking, a set of FDs satisfies split-freeness if it does not imply two FDs where R.H.S. of each of these two FDs is contained in the L.H.S. of the other FD.

### 2.10.3 Intersection

Two nontrivial FDs of the forms \(R:X \rightarrow A\) and \(R:Y \rightarrow A\), where \(R\) is a relation schema, \(X,Y\) are sets of attributes of \(R\), and \(A,B\) are two attributes of \(R\), are said to be incomparable if \(X\) and \(Y\) are incomparable. Two sets are incomparable if either is a subset of the other.

**Definition 2.10.4** A set of FDs \(F\) over \(R\) satisfies the intersection property if \(\forall F_j \in F, \forall A \in R_j\), whenever there exist incomparable FDs, \(R_j:X \rightarrow A, R_j:Y \rightarrow A \in F_j^+\), then \(R_j:X \cap Y \rightarrow A \in F_j^+\).

Intuitively, if a set of FDs satisfies the intersection property if it does not imply two non-trivial FDs that imply the same attribute.
2.10.4 Reduced set of FDs and INDs

**Definition 2.10.5** The projection of a set of FDs \( F_i \) over relation schema \( R_i \) onto a set of attributes \( Y \subseteq R_i \), denoted by \( F_i[Y] \), is given by \( F_i[Y] = \{ R_i : W \rightarrow Z \mid R_i : W \rightarrow Z \in F_i^+ \text{ and } WZ \subseteq Y \} \).

**Definition 2.10.6** A set of attributes \( Y \subseteq R_i \) is said to be reduced with respect to \( R_i \) and a set of FDs \( F_i \) over \( R_i \) (or simply reduced with respect to \( F_i \) if \( R_i \) is understood from context) if \( F_i[Y] \) contains only trivial FDs. A set of FDs and INDs \( \Sigma = F \cup I \) is said to be reduced if \( \forall R_i[X] \subseteq R_j[Y] \in I, Y \) is reduced with respect to \( F_j \).

We illustrate the reduced property with an example. Our example consists of two relation schemas \( \text{author}[\text{name}, \text{DOB}] \) and \( \text{person}[\text{name}, \text{DOB}] \), an IND \( \sigma_I = \text{author}[\text{name}, \text{DOB}] \subseteq \text{person}[\text{name}, \text{DOB}] \), and an FD \( \sigma_F = \text{person} : \text{name} \rightarrow \text{DOB} \).

\( \sigma_I \) guarantees that any tuples in \( \text{author} \) must have an equivalent tuple in \( \text{person} \). \( \sigma_F \) guarantees \( \text{name} \) imply \( \text{DOB} \)s in such tuples in \( \text{person} \). This in turn guarantees \( \text{name} \) imply \( \text{DOB} \)s in all tuples in \( \text{author} \).

2.10.5 Theorem that Guarantees Non-Interaction between FD and IND

Taken from [11], the following theorem states the conditions needed to prevent interaction:

**Theorem 2.10.7** If \( I \) (a set of INDs) is proper circular, \( F \) (a set of FDs) satisfies either the intersection property or the split-freeness property and \( \Sigma = (F \cup I) \) is reduced then \( F \) and \( I \) do not interact.
Chapter 3

Efficient XML Constraint Implication

In order to achieve efficient XML FD and IND implication, we first reduce XML FD and IND implication to relational FD and IND implication in Section 3.1. However, Chandra et al. [8] proved relational FD and IND implication is undecidable. Therefore, in Section 3.2, we introduce conditions on XML FDs and INDs that ensure the transformed relational FDs and INDs do not interact.

3.1 Reduction of XML constraint implication to relational constraint implication

Relational constraint reasoning is a well researched area. Being able to reduce XML constraints to their relational counterparts can resolve a lot of uncertainties involving XML constraints.

In this section, we aim to reduce the XML FD and IND implication problem to the relational FD and IND implication problem. We specify the problem of XML FD and IND implication here:

**Problem Statement** Let $\Delta$ be a DTD, $\Sigma_\Delta$ be a set of FDs and INDs on $\Delta$, and $\sigma_\Delta$ be an FD or IND on $\Delta$. Determine if $\Sigma_\Delta \models \sigma_\Delta$.

We first construct a schema transformation function $\mu_{schema}$ that takes $\Delta$ and
outputs a set of relation schemas $R$, denoted:

$$R = \mu_{\text{schema}}(\Delta)$$

In Section 3.1.3, we will define and explain this schema transformation function.

Next we construct a constraint transformation function $\mu_{\text{constraint}}$ takes $\sigma_\Delta$ and produces $\sigma_R$ that conforms to $R$, denoted:

$$\sigma_R = \mu_{\text{constraint}}(\sigma_\Delta)$$

We will define and explain this constraint transformation function in more detail in Section 3.1.4.

When given a set of XML constraints $\Sigma_\Delta$ on $\Delta$, we define:

$$\mu_{\text{constraints}}(\Sigma_\Delta) = \{ \mu_{\text{constraint}}(\sigma_\Delta) \mid \sigma_\Delta \in \Sigma_\Delta \}$$

$\mu_{\text{constraints}}$ repeatedly calls $\mu_{\text{constraint}}$ on each constraint in $\Sigma_\Delta$, and outputs the set of relational constraints.

We also develop a data transformation function $\mu_{\text{data}}$ that takes a DTD $\Delta$ and an XML database $D_\Delta$ that conforms to $\Delta$, and output a relational database $D_R$ that conforms to $R = \mu_{\text{schema}}(\Delta)$, denoted:

$$D_R = \mu_{\text{data}}(\Delta, D_\Delta)$$

$\mu_{\text{data}}$ will be used in our proof that XML constraint implication can be reduced to relational constraint implication. We will construct $\mu_{\text{data}}$ in Section 3.1.5.

For simplicity, we overload the $\mu$ operator so that $\mu$ is $\mu_{\text{schema}}$ when the argument is a DTD, $\mu_{\text{constraint}}$ if the argument is an XML FD or IND, $\mu_{\text{constraints}}$ if the argument is a set of XML constraints, and $\mu_{\text{data}}$ if the argument is a DTD and XML database.

We then prove that XML constraint implication can be reduced to relational constraint implication in Section 3.1.7. After proving the reduction is correct, we then present an algorithm to solve XML constraint implication with relational constraint implication in Section 3.1.8.
3.1.1 Assumptions

1. DTDs are simple DTDs as defined in Section 2.2: choice-free, duplicate-free, and acyclic.

2. For simplicity, we assume for all DTDs \( \Delta = (E,A,\Phi,\Pi,r) \) the DTD group including root \( r \) has no attributes.

3. All the XML INDs are typed.

4. Since all XML INDs are typed, Definition 2.9.1 guarantees all attributes within a DTD group are unique. Therefore, for simplicity, we can assume without loss of generality that each DTD group has only one element - the group node.

5. All the leaf nodes are attributes.

3.1.2 XML Database Equivalence (\( \equiv \))

To simplify the XML model for constraint reasoning, we ignore order of elements. Thus we need a database equivalence operator (\( \equiv \)) that determines two XML databases \( D_1 = (V_1,lab_1,ele_1,att_1,root_1) \) on DTD \( \Delta_1 = (E_1,A_1,\Phi_1,\Pi_1,r_1) \) and \( D_2 = (V_2,lab_2,ele_2,att_2,root_2) \) on DTD \( \Delta_2 = (E_2,A_2,\Phi_2,\Pi_2,r_2) \) to be equivalent if their content is the same but their order of elements may be different.

Here we define XML database equivalence (\( \equiv \)):

Two XML elements \( x \in V_1, y \in V_2; x \equiv y \) iff:

1. node-name(\( x \)) = node-name(\( y \))

2. \( \forall l \in \Pi_1(lab_1(x)), \exists l \in \Pi_2(lab_2(y)), \text{s.t. } att_1(x,l) = att_2(y,l) \)

3. \( \forall l \in \Pi_2(lab_2(y)), \exists l \in \Pi_1(lab_1(x)), \text{s.t. } att_2(y,l) = att_1(x,l) \)

4. \( \forall x' \in child(x), \exists y' \in child(y) \text{s.t. } x' \equiv y' \)

5. \( \forall y' \in child(y), \exists x' \in child(x) \text{s.t. } y' \equiv x' \)

27
Figure 3.1: Equivalent UBCBookstore Instance Example

$D_1 \equiv D_2$, iff $\text{root}_1 \equiv \text{root}_2$.

For example, the XML database in Figure 1.2 and the one in Figure 3.1 are equivalent. They differ by the order of the two books. The two databases would both satisfy or both not satisfy any set of FDs and INDs.

XML FDs and INDs do not put constraints on the order of data; therefore, we can derive the following Lemma:

Lemma 3.1.1 For any two XML databases $D_1$ and $D_2$, if $D_1 \equiv D_2$, then $\forall$ XML FDs or INDs $\sigma$, $D_1 \models \sigma$ iff $D_2 \models \sigma$.

Proof The $\equiv$ operator is symmetric; therefore, we only need to prove one direction of 'if and only if'. We will prove by contradiction.

For the proof on FDs, let the constraint $\sigma$ be an FD. Assume $D_1 \not\models \sigma$ but $D_2 \models \sigma$. $D_1$ must contain two elements $x, y$ with the same L.H.S. attributes of $\sigma$, but they have different R.H.S. attribute of $\sigma$. Since $D_2 \equiv D_1$, $D_2$ must contain two elements $x', y'$ s.t. $x' \equiv x$ and $y' \equiv y$. As a result, $x'$ contains the same set of attributes as $x$, and $y'$ contains the same set of attributes as $y$. $x'$ and $y'$ would cause $D_2$ to violate $\sigma$. This violates the hypothesis; thus Lemma 3.1.1 holds for FDs.

For the proof on INDs, let the constraint $\sigma$ be an IND. Assume $D_1 \not\models \sigma$ but $D_2 \models \sigma$. $D_1$ must contain an element $x$ whose attributes fit the L.H.S. of $\sigma$, but $D_1$ must not contain an element $y$ that satisfies R.H.S. of $\sigma$ with respect to $x$. Since $D_1 \equiv D_2$, $D_2$ must contain $x'$ s.t. $x' \equiv x$. Since $D_2 \models \sigma$, $D_2$ must contain $y'$ s.t. $y'$ satisfies R.H.S. of $\sigma$ with respect to $x'$. Again because $D_1 \equiv D_2$, $D_1$ must contain an element that is equivalent to $y'$, such an element would satisfy R.H.S. of $\sigma$ with respect to $x$. This violates the hypothesis; thus Lemma 3.1.1 holds for INDs too.
3.1.3 Schema Transformation Function — \( \mu(\Delta) \)

The XML schema transformation function \( \mu(\Delta) \) in Figure 3.2 is designed to translate a DTD \( \Delta \) into a set of relation schemas \( R \) that retains the DTD group structure of \( \Delta \). We

Input: DTD \( \Delta = (E, \Phi, \Pi, r) \)
Output: A set of Relation Schemas \( R \)

1. \( R = \{ \} \)
2. For each group \( G_i \in E, G_i \neq r \)
3. create a new relation schema \( R_i \)
4. create a new field for each attribute in \( G_i \), and
   label it with the \( SP \) of the attribute
5. create a new field ‘id’
6. create a new field ‘pid’
7. \( R = R \cup \{R_i\} \)

**Figure 3.2: Schema Transformation Algorithm — \( \mu(\Delta) \)**

In Line 3, we create a new relation \( R_i \) for each DTD group \( G_i \) because we see each DTD group as a schema describing an entity. All the attributes within a DTD group are the properties of the entity that this group is describing; therefore, we create a new field for each attribute within \( G_i \). We then create the fields ‘id’ and ‘pid’ to retain the parent-child relation among DTD groups. Such relations will be expressed as INDs as transformation generated constraints.

Running this algorithm against our example DTD \( \Delta_{UBCBookstore} \) in Figure 1.1, we will generate three relation schemas:

- subject[id,pid,name]
- person[id,pid,name,DOB]

They correspond to the three DTD groups in \( \Delta_{UBCBookstore} \). Every attribute in a DTD group becomes a field in the corresponding relation; in addition, ‘id’ and ‘pid’ are present in every relation to retain the parent-child relation of DTD groups.
Transformation Generated Constraints — $\Psi(\Delta)$

During schema transformation, parent-child relations can be retained by generating an IND for each child DTD group whose 'pid' is subsumed by its parent's 'id'. In addition, 'id' is the node identifying attribute; thus it implies all the other attributes within a DTD group.

Input: $\text{DTD } \Delta = (E, \Gamma, \Phi, \Pi, r)$
Output: A set of INDs $\Psi(\Delta)$

1. $\Psi(\Delta) = \{\}$
2. For each DTD Group $G_i \in E$
3. if $\text{parent}(G_i) \neq \emptyset$
4. $\Psi(\Delta) = \Psi(\Delta) \cup \{R_i[\text{pid}] \subseteq \text{parent}(R_i)[\text{id}]\}$
5. For each DTD Attribute $A \in \Pi(G_i)$
6. $\Psi(\Delta) = \Psi(\Delta) \cup \{R_i : \text{id} \rightarrow A\}$

Figure 3.3: Creating Transformation Generated Constraints Algorithm — $\Psi(\Delta)$

We can apply $\Psi$ to our example DTD $\Delta_{UBCBookstore}$ in Figure 1.1, and will result in the transformation generated constraints in Table 3.1. The field 'id' in each relation implies every other attribute within the relation. In addition, book is a child group of subject, and we have the IND $\text{book[pid]} \subseteq \text{subject[id]}$ that captures the parent-child relation.

3.1.4 Constraint Transformation Function — $\mu(\sigma_\Delta)$

Constraint transformation function $\mu(\sigma_\Delta)$ takes an XML constraint of either FD $F_\Delta$, or IND $I_\Delta$, and translates it to relational constraint $F_R$ (Figure 3.4) or $I_R$ (Figure 3.5) respectively:

For example, the FD $\text{BookFD#1}$ on DTD group $\text{person}$ in Table 1.1 is:

$\text{UBCBookstore.person} : \text{name} \rightarrow \text{DOB}$
Given an XML FD $F_D: G_k : SP_{k,1}, \ldots, SP_{k,n} \rightarrow SP_{k,u}$,

$$\mu(F_D) = R_k : SP_{k,1}, \ldots, SP_{k,n} \rightarrow SP_{k,u}$$

**Figure 3.4:** FD transformation Algorithm — $\mu(FD)$

$\mu(BookFD#1)$ is on the transformed relation $person$:

$$person : name \rightarrow DOB$$

For another example, the IND $BookIND#1$ on DTD groups $book$ and $person$ in Table 1.1 is:

$$UBCBookstore.subject.book.author[name, DOB]$$

Given an XML IND $I_\Delta$: $G_k [SP_{k,1}, \ldots, SP_{k,n}] \subseteq G_t [SP_{t,1}, \ldots, SP_{t,n}]$,

$$\mu(I_\Delta) = R_k [SP_{k,1}, \ldots, SP_{k,n}] \subseteq R_t [SP_{t,1}, \ldots, SP_{t,n}]$$

**Figure 3.5:** IND transformation Algorithm — $\mu(IND)$
Input: DTD $\Delta = (E, A, \Phi, \Pi, r)$, and a set of XML constraints $\Sigma_\Delta$ on $\Delta$
Output: A set of FDs + INDs $\mu(\Sigma_\Delta)$

1. $\mu(\Sigma_\Delta) = \{\}$
2. For each $\sigma_\Delta \in \Sigma_\Delta$
3. $\mu(\Sigma_\Delta) = \mu(\Sigma_\Delta) \cup \mu(\sigma_\Delta)$

**Figure 3.6: Constraints Transformation Algorithm — $\mu(\Sigma_\Delta)$**

$\subseteq UBCBookstore.person[name, DOB]$

$\mu(\text{BookIND#1})$ is on the transformed relations book and person:

$\text{book}[\text{author.name}, \text{author.DOB}] \subseteq \text{person}[\text{name}, \text{DOB}]$

The constraint transformation function $\mu$ defined in this section takes one XML constraint $\sigma$ at a time. We are going to overload the function $\mu$ so that it takes a set of XML constraints $\Sigma$ as argument. The function $\mu(\Sigma)$ calls $\mu(\sigma)$ for each $\sigma \in \Sigma$, and returns the results as a set.

In Figure 3.6, we also define $\mu(\Sigma_\Delta)$ that transforms a set of constraints on $\Delta$. It transforms each $\sigma_\Delta \in \Sigma_\Delta$ by $\mu(\sigma_\Delta)$ in Figure 3.4 and Figure 3.5, then returns the set of resulting constraints.

If we apply this constraints transformation function to FDs and INDs in UBC-Bookstore Table 1.1, it will result in the set of constraints in Table 3.2.

### 3.1.5 Data Transformation Functions — $\mu(D_\Delta), \mu^{-1}(D_R)$

As shown in Figure 3.7, the data transformation function $\mu$ takes an XML database $D_\Delta$ that conforms to DTD $\Delta$ and produces a relational database $D_R$ that conforms to the transformed relational schema $\mu(\Delta)$. In Lines 2,3, the algorithm creates a new tuple for every group element in $D_\Delta$. It creates a unique ID for the tuple and put it in the field ‘id’ in Line 4; and put the group element’s parent ID into ‘pid’ in Line 6. This retains the parent-child relation among the group elements. In Lines 8-11, it puts all attributes in the group in their corresponding fields in the tuple, and puts NULL in the fields for the attributes that do not exist.
Table 3.2: $\mu(\Sigma_{\text{UBCBookstore}})$

<table>
<thead>
<tr>
<th>Input: DTDA = (E, A, $\Phi, \Pi, r$), XML database $D_\Delta$ = (V, lab, ele, att, root)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output: Relational database $D_R$</td>
</tr>
<tr>
<td>1. create tables $R_1, \ldots, R_n$ for $D_R$ according to $\mu(\Delta)$, where $n =</td>
</tr>
<tr>
<td>2. for each group node $e \in V$, where $\text{lab}(e) = G_i$, and $G_i \neq r$</td>
</tr>
<tr>
<td>3. create a new tuple $t$ in table $R_i$</td>
</tr>
<tr>
<td>4. create a unique ID UID$(e)$ and put it in $t[\text{id}]$</td>
</tr>
<tr>
<td>5. if $e$ has a parent node parent$(e)$ and parent$(e) \neq r$</td>
</tr>
<tr>
<td>6. $t[\text{pid}] = \text{UID}$(parent$(e)$)</td>
</tr>
<tr>
<td>7. else $t[\text{pid}] = \text{NULL}$</td>
</tr>
<tr>
<td>8. for each attribute $A \in \Pi(\text{lab}(e))$</td>
</tr>
<tr>
<td>9. if $e.A$ does not exist</td>
</tr>
<tr>
<td>10. then $t[A] = \text{NULL}$</td>
</tr>
<tr>
<td>11. else $t[A] = e.A$</td>
</tr>
</tbody>
</table>

Figure 3.7: Data Transformation Algorithm

For simplicity, if $\Delta$ is understood, we shorten the notation from $\mu(\Delta, D_\Delta)$ to $\mu(D_\Delta)$ for the rest of this document.

We demonstrate the data transformation on the XML database $D_{\text{UBCBookstore}}$ (Figure 1.2). After translation, we get the three tables in Table 3.3. In Figure 1.2, there are two book instances with isbn ‘001’ and ‘002’. $\mu(D_{\text{UBCBookstore}})$ put these two books into two different tuples in the table book. Notice each tuple has a unique
ID in its ‘id’ field. The two books in Figure 1.2 are under a subject node, which has a name attribute ‘DB’. The two corresponding book tuples have the unique ID of the subject tuple in their pid fields; therefore, it keeps record of the parent-child relation among groups.

<table>
<thead>
<tr>
<th>id</th>
<th>pid</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID1</td>
<td>NULL</td>
<td>DB</td>
</tr>
</tbody>
</table>

(a) Translated Table ‘subject’

<table>
<thead>
<tr>
<th>id</th>
<th>pid</th>
<th>isbn</th>
<th>year</th>
<th>title</th>
<th>price</th>
<th>#pages</th>
<th>author.name</th>
<th>author.DOB</th>
<th>pub</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID2</td>
<td>ID1</td>
<td>001</td>
<td>2001</td>
<td>Survive</td>
<td>120</td>
<td>550</td>
<td>Smith</td>
<td>19490101</td>
<td>FreePress</td>
</tr>
<tr>
<td>ID3</td>
<td>ID1</td>
<td>002</td>
<td>2001</td>
<td>DB</td>
<td>70</td>
<td>320</td>
<td>Smith</td>
<td>19490101</td>
<td>FreePress</td>
</tr>
</tbody>
</table>

(b) Translated Table ‘book’

<table>
<thead>
<tr>
<th>id</th>
<th>pid</th>
<th>name</th>
<th>DOB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID4</td>
<td>NULL</td>
<td>Smith</td>
<td>19490101</td>
</tr>
</tbody>
</table>

(c) Translated Table ‘person’

Table 3.3: \( \mu(D_{UBCBookstore}) \)

On the other hand, in Figure 3.8, data transformation function \( \mu^{-1} \) takes a relational database \( D_R \) that conforms to relational schema \( \mu(\Delta) \), and produces an XML database \( D_\Delta \) that conforms to \( \Delta \).

For simplicity, if \( \Delta \) is understood, we shorten the notation from \( \mu^{-1}(\Delta, D_R) \) to \( \mu(D_R) \) for the rest of this document.

The reverse data transformation algorithm \( \mu^{-1} \) in Figure 3.8 does the reverse of \( \mu \) in Figure 3.7. Given output from \( \mu \), it produces XML databases that are equivalent (\( \equiv \)) to the input of \( \mu \). Studying the two algorithms, we can easily derive the following propositions:

**Proposition 3.1.2** For all XML databases \( D_\Delta \) on DTD \( D_\Delta \)

\[ D_\Delta \equiv \mu^{-1}(\mu(D_\Delta)) \]
Input: a DTD $\Delta$, a Relational Instance $D_R$
Output: XML database $D_{\Delta}$ that conforms to $\Delta$

1. create an XML node root
2. for each table $R_i \in \mu(\Delta)$
3. for each tuple $t_j \in R_i$
4. create an XML element $e_{t_j[id],t_j[pid]}$
5. for each attribute $A \in t_j$
6. if $t_j.A \neq \text{NULL}$
7. $e_{t_j[id],t_j[pid]}[A] = t_j.A$
8. for each $e_{id,pid}$ created above
9. if $pid = \text{NULL}$
10. then make $e_{id,pid}$ a child of root
11. else make $e_{id,pid}$ a child of some $e'_{pid,*}$

Figure 3.8: Reverse Data Transformation Algorithm

Proposition 3.1.3 $\forall$ relational databases $D_R$ on a set of relation schemas $R$

$$D_R \equiv \mu(\mu^{-1}(D_R))$$

3.1.6 Correctness of XML Constraint Propagation

With the constraint transformation functions defined in Section 3.1.4, we first show the preservation of FDs and INDs of the constraint transformation functions both from XML constraints. Then using this result, we prove the sound and completeness of using relational constraint implication to imply XML constraints.

Preservation of FDs

We first claim that any XML FD is preserved by the constraint transformation. We show our claim must be correct by contradiction. If an FD $F$ were violated after propagation, there must exist two tuples in the transformed database $\mu(D_\Delta)$ with the same set of L.H.S. attributes of the transformed FD $\mu(F)$, but different R.H.S.
attributes. Since every tuple in $\mu(D_\Delta)$ is traced from a group in the XML Database $D_\Delta$ on $\Delta$ that conforms to $F$, there must exist two group instances in $D_\Delta$ that have the same L.H.S. attributes of $F$, but different R.H.S. attributes. This would violate the assumption that $D_\Delta$ conforms to $F$; thus, our claim holds.

**Lemma 3.1.4** \( \forall \) XML databases \( D_\Delta = \{V, \text{lab, ele, att, root}\} \) that conforms to DTD \( \Delta \), \( \forall \) XML FD \( F \), if \( D_\Delta \models F \), then \( \mu(\Delta, D_\Delta) \models \mu(F) \).

**Proof** \( F \) has the form: \( \tau_k : SP_{k,1} \ldots SP_{k,n} \rightarrow SP_{k,u} \). If \( D_\Delta \models F \), then \( \forall x, y \in V \), where \( \text{lab}(x) = \text{lab}(y) = \tau_k \), and \( \bigwedge_{i \in 1, \ldots, n} x.SP_{k,i} = y.SP_{k,i} \), then \( x.SP_{k,u} = y.SP_{k,u} \).

Meanwhile, \( \mu(F) = R_k[SP_{k,1}, \ldots, SP_{k,n}] \rightarrow SP_{k,u} \). If our claim were wrong, there would exist \( t_1, t_2 \in R_k(D_\Delta) \) such that \( \bigwedge_{i \in 1, \ldots, n} t_1.SP_{k,i} = t_2.SP_{k,i} \), and \( t_1.SP_{k,u} \neq t_2.SP_{k,u} \). Since \( \mu(D_\Delta) \) is transformed from \( D_\Delta \), the tuple \( t_1 \), and \( t_2 \) must have their respective corresponding group nodes \( x, y \) such that \( \bigwedge_{i \in 1, \ldots, n, u} t_1.SP_{k,i} = x.SP_{k,i} \) and \( t_2.SP_{k,i} = y.SP_{k,i} \). This shows \( \exists x, y \in V \), where \( \text{lab}(x) = \text{lab}(y) = \tau_k \), and \( \bigwedge_{i \in 1, \ldots, n} x.SP_{k,i} = y.SP_{k,i} \), but \( x.SP_{k,u} \neq y.SP_{k,u} \). Thus it contradicts \( D_\Delta \models F \), and our claim holds.

Next we show that the reverse transformation preserves FDs.

We prove by contradiction. If an FD \( F \) were violated after reverse propagation, there must exist two group nodes \( x, y \) in the reverse transformed database \( \mu^{-1}(D_R) \) with the same set of L.H.S. attributes of the \( F \), but different R.H.S. attributes. Since every group in \( \mu^{-1}(D_R) \) is traced from a tuple in \( D_R \) on that conforms to \( \mu(F) \), there must exist two tuples in \( D_R \) that have the same L.H.S. attributes of \( \mu(F) \), but different R.H.S. attributes. This would violate the assumption that \( D_R \) conforms to \( \mu(F) \); thus our claim holds.

**Lemma 3.1.5** \( \forall \) relational databases \( D_R \) conforms to the set of relation schemas \( \mu(\Delta) \), where \( \Delta \) is a DTD, \( \forall \) XML FDs \( F \), if \( D_R \models F \), then \( \mu^{-1}(D_R) \models F \).

**Proof** Let \( F \) be \( \tau_k : SP_{k,1} \ldots SP_{k,n} \rightarrow SP_{k,u} \). Then \( \mu(F) \) is \( R_k[SP_{k,1}, \ldots, SP_{k,n}] \rightarrow SP_{k,u} \).

If \( D_R \models \mu(F) \), then \( \forall t_1, t_2 \in R_k(D_R) \), if \( \bigwedge_{i \in 1, \ldots, n} t_1.SP_{k,i} = t_2.SP_{k,i} \), then \( t_1.SP_{k,u} = t_2.SP_{k,u} \).
Let $\mu^{-1}(D_R) = (V, \text{lab}, \text{ele}, \text{att}, \text{root})$. However, if our claim were wrong, there would exist $x, y \in V$, where $\text{lab}(x) = \text{lab}(y) = \tau_k$, and $\bigwedge_{i \in \{1, \ldots, n\}} x.SP_{k,i} = y.SP_{k,i}$, but $x.SP_{k,u} \neq y.SP_{k,u}$. Since all group nodes in $\mu^{-1}(D_R)$ are transformed from $D_R$, the group nodes $x, y$ must have their respective tuples $t_1, t_2$ such that $\bigwedge_{i \in \{1, \ldots, n\}} t_1.SP_{k,i} = x.SP_{k,i}$ and $t_2.SP_{k,i} = y.SP_{k,i}$. This shows $\exists t_1, t_2 \in R_k(D_R)$, and $\bigwedge_{i \in \{1, \ldots, n\}} t_1.SP_{k,i} = t_2.SP_{k,i}$, but $t_1.SP_{k,u} \neq t_2.SP_{k,u}$. Thus it contradicts $D_R \models \mu(F)$, and our claim holds.

**Preservation of INDs**

We then claim that every XML IND is preserved by the constraint transformation. Data transformation transforms every group in an XML database $D_\Delta$ to a tuple in the transformed relational database $\mu(D_\Delta)$. In addition, every tuple in $\mu(D_\Delta)$ comes from a group in $D_\Delta$. Therefore, if $D_\Delta$ satisfies an XML IND $I$, then every group that appears on the L.H.S. of $I$ must have a corresponding group on the R.H.S. of $I$. All of these groups and only these groups are transformed to L.H.S. and R.H.S. tables of $\mu(I)$ respectively. Therefore, $\mu(D_\Delta)$ satisfies $\mu(I)$.

**Lemma 3.1.6** \forall XML databases $D_\Delta = \{V, \text{lab}, \text{ele}, \text{att}, \text{root}\}$ that conforms to DTD $\Delta$, \forall XML INDs $I$, if $D_\Delta \models I$, then $\mu(D_\Delta) \models I$.

**Proof** I has the form: $\tau_i[SP_{k,1}, \ldots, SP_{k,n}] \subseteq \tau_i[SP_{l,1}, \ldots, SP_{l,n}]$. If $D_\Delta \models I$, then $\forall x \in V$, where $\text{lab}(x) = \tau_i$, $\exists y \in V$, of which $\text{lab}(y) = \tau_i$, and $\bigwedge_{i \in \{1, \ldots, n\}} x.SP_{k,i} = y.SP_{i,j}.$

Meanwhile, $\mu(I) = R_k[SP_{k,1}, \ldots, SP_{k,n}] \subseteq R_l[SP_{l,1}, \ldots, SP_{l,n}]$. And $\mu(D_\Delta)$ ensures every group node $z \in V$ of group type $\tau_i$ is transformed into a corresponding tuple $t$ in $R_i(\mu(D_\Delta))$ with all the attributes of $z$ put into the respective fields in $t$, and every $t$ in $R_i(\mu(D_\Delta))$ is traced from some group node $z \in V$ of group type $\tau_i$. Therefore, every $t_1 \in R_k(\mu(D_\Delta))$ is traced from an $x \in V$, where $\text{lab}(x) = \tau_k$. Together with the fact that every $y \in V$, where $\text{lab}(y) = \tau_i$, is transformed into $R_i(\mu(D_\Delta))$, $\mu(D_\Delta) \models \mu(I)$ is satisfied.

Next we show that the reverse transformation preserves INDs. Reverse data transformation transforms every tuple in a relational database $D_R$ to a group in the
reverse transformed XML database $\mu^{-1}(D_R)$. In addition, every group in $\mu^{-1}(D_R)$ comes from a tuple in $D_R$. Therefore, if $D_R$ satisfies a transformed IND $\mu(I)$, then every tuple that appears on the L.H.S. of $\mu(I)$ must have a corresponding tuple on the R.H.S. of $\mu(I)$ to. All of these tuples and only these tuples are reverse transformed to L.H.S. and R.H.S. groups of $I$ respectively. Therefore, $\mu^{-1}(D_R)$ satisfies $I$.

Lemma 3.1.7 \forall relational databases $D_R$ that conforms to relational schema $\mu(\Delta)$, \forall XML INDs $I$, if $D_R \models \mu(I)$, then $\mu^{-1}(D_R) \models I$.

Proof Let $I$ be $\tau_k[SP_{k,1}, \ldots, SP_{k,n}] \subseteq \tau_I[SP_{I,1}, \ldots, SP_{I,n}]$. Then $\mu(I) = R_k[SP_{k,1}, \ldots, SP_{k,n}] \subseteq R_I[SP_{I,1}, \ldots, SP_{I,n}]$.

If $D_R \models \mu(I)$, then $\exists t_1 \in R_k(D_R)$, $\exists t_2 \in R_I(D_R)$, such that $\land_{i=1, \ldots, n} t_1.SP_{k,i} = t_2.SP_{I,i}$.

Let $\mu^{-1}(D_R) = (V, \text{lab}, \text{ele}, \text{att}, \text{root})$. Every group node $z$ in $\mu^{-1}(D_R)$ has a corresponding tuple $t$ in $D_R$ with all values of attributes of $z$ coming from their respective fields in $t$. Thus every $x \in V$, where $\text{lab}(x) = \tau_k$ has an corresponding $t_1 \in R_k(D_R)$ with same attribute/field values. Together with the fact that $D_R \models \mu(I)$, and every $t_2 \in R_I(D_R)$ has a corresponding $y \in V$, where $\text{lab}(y) = \tau_I$, $\mu^{-1}(D_R) \models I$ is satisfied.

3.1.7 XML to Relational Constraint Implication Reduction

We can use relational constraint implication to test implication of XML constraints:

Theorem 3.1.8 Given a set of XML constraints $\Sigma$ and an XML constraint $\sigma$, $\Sigma \models \sigma$ iff $\mu(\Sigma) \models \mu(\sigma)$

To prove Theorem 3.1.8, we break it into two lemmas. Lemma 3.1.9 is responsible for the ‘if’ half of Theorem 3.1.8, and we prove Lemma 3.1.9 below. Lemma 3.1.10 is responsible for the ‘only if’ half of Theorem 3.1.8, and we prove Lemma 3.1.10 at the end.

Lemma 3.1.9 Given a set of XML constraints $\Sigma$ and an XML constraint $\sigma$, if $\Sigma \models \sigma$, then $\mu(\Sigma) \models \mu(\sigma)$
Algorithm TestXMLImplication(Σ, σ)

Input: a set of XML FDs and INDS Σ, an FD or IND σ
Output: True if Σ |= σ, false otherwise
1. return μ(Σ) |= μ(σ)

Figure 3.9: Algorithm Testing XML Constraint Implication

Proof For every relational database Dr, Lemma 3.1.5 and Lemma 3.1.7 prove μ⁻¹(Dr) |= Σ if Dr |= μ(Σ).
If Σ |= σ, then μ⁻¹(Dr) |= σ. By Lemma 3.1.4 and Lemma 3.1.6, μ(μ⁻¹(Dr)) = μ(σ). By Proposition 3.1.3, Dr ≡ μ⁻¹(Dr). By Lemma 3.1.1, Dr |= μ(σ).
Therefore, μ(Σ) |= μ(σ) □

Lemma 3.1.10 Given a set of XML constraints Σ and an XML constraint σ, if μ(Σ) |= μ(σ), then Σ |= σ

Proof For every XML database DΔ, Lemma 3.1.4 and Lemma 3.1.6 prove μ(DΔ) |= μ(Σ) if DΔ |= Σ.
If μ(Σ) |= μ(σ), then μ(DΔ) |= μ(σ). By Lemma 3.1.5 and Lemma 3.1.7, μ⁻¹(μ(DΔ)) |= σ. By Proposition 3.1.2, DΔ ≡ μ⁻¹(μ(DΔ)). By Lemma 3.1.1, DΔ |= σ. Therefore, Σ |= σ □

3.1.8 Algorithm to Test XML Constraint implication by Relational Constraint Implication

From the sections above, we have proved that testing XML constraint implication can be handled by testing relational constraint implication. We build an algorithm here showing the steps required to employ testing relational constraint implication to solve the XML implication problem. The algorithm is very simple, it transforms a set of XML constraints and a single XML constraint to relational constraints, then test if the set of transformed relational constraints implies the one transformed relational constraint.
3.2 Guaranteeing No Interaction Between XML FDs and INDs

Now that we can use relational implication to test XML constraint implication, we can concentrate on complexity. Detecting XML constraint violation, and translating XML constraints via mappings all require an efficient XML constraint implication algorithm.

XML constraint implication is undecidable because relational FD + IND implication is undecidable [13],[8]. As we proved in Section 3.1.6 we can use relational FD + IND implication to solve XML FD + IND implication, we inherit the undecidability property.

The undecidability complexity stems from interaction of FDs and INDs. However, according to Theorem 2.10.7 non-interaction of relational FDs and INDs is possible. Building on it, this section aims to achieve non-interaction of XML FDs and INDs.

In order for XML FDs and INDs to not interact, after transformation of XML constraints to relational constraints, the relational constraints must satisfy conditions in Theorem 2.10.7.

For the rest of this section, we will introduce the necessary restrictions on our XML constraints such that after the FDs and INDs are transformed to their relational counterparts, they do not interact.

3.2.1 Retaining Proper Circular Property

Proper Circular Property, defined in Definition 2.10.2, is one of the conditions of Theorem 2.10.7 necessary to avoid FD and IND interaction. Being able to retain proper circular property during transformation is necessary for efficient constraint implication.

In order to retain the proper circular property of XML INDs, we deal with the slightly simpler retaining of acyclic XML INDs first. Extending on that result, we then explain how to retain the proper circular property.
Retaining Acyclic Property

Intuitively, each IND directionally connects two DTD groups. Two INDs can chain together to form a longer path if the DTD group on the R.H.S. of one IND is the DTD group on the L.H.S. of the other IND. For example, if the first IND connects DTD group $G_1$ to $G_2$, and second IND connects $G_2$ to $G_3$, then there is a path from $G_1$ to $G_3$. A path becomes a cycle when a DTD group appears in a path more than once. For example, if we add a third IND that connects DTD group $G_3$ to $G_1$ to the two INDs above, we will form a cycle.

Transforming a set of XML INDs with cycles to relational INDs will definitely violate the acyclic property (Definition 2.10.1). Consequently, FDs and INDs will interact, and we cannot efficiently test constraint implication. In addition, transformation created INDs involving ‘id’ and ‘pid’ may complete cycles that do not exist before; therefore, simply avoiding cycles in the set of XML INDs is not enough.

In Figure 3.10a, IND $G_i[X] \subseteq G_j[Y]$ is demonstrated by an arrow from $G_i$ to $G_j$. The INDs in Figure 3.10a is transformed to Figure 3.10b with each $R_i$ corresponds to $G_i$. As shown in Figure 3.10b, the transformed INDs complete a cycle after the addition of transformation generated INDs.

Our digraph has leaf DTD groups, DTD groups with no children, as vertices demonstrated by encircling ellipses. Arrows between vertices if there are INDs among ancestors of the 2 leaf nodes. Cycles in this digraph illustrate possible cy-
cles in the relational INDs, and the cycles are complete if within each connected vertex, the group pointed to by the arrow head (incoming IND target) is a descendant of the group that the outgoing arrow (outgoing IND source) points away. This is because after transformation, there will be INDs from descendants to ancestors, thereby completing circles by linking arrows within each vertex.

Below, we give a set of formal conditions for a set of INDs to retain the acyclic property after transformation.

Given a DTD $\Delta = (E, A, \Phi, \Pi, r)$, and a set of INDs $I$.

$I \cup \{G_i[pid] \subseteq G_2[id] \mid G_1, G_2 \in E \text{ and } G_2 = \text{parent}(G_1)\}$ has cycles if:

1. $\exists \sigma \in I \text{ s.t. } \sigma: G_1[X] \subseteq G_2[Y]$, where $G_1 \in \text{ancestor}\_or\_self(G_2)$, or
2. We construct a digraph where leaf DTD groups, DTD groups with no children, make up the vertices. The directed edges are:

   \[
   \{ (V_1, V_2) \mid G_1[X] \subseteq G_2[Y] \in I, \text{ where } G_1, G_2, V_1, V_2 \in E, V_1, V_2 \text{ are leaves, and } G_1 \in \text{ancestors}\_or\_self(V_1), G_2 \in \text{ancestors}\_or\_self(V_2) \}
   \]

   If a cycle $C$ exists in this digraph, we call $I$ circular if

   for each $V_i$ in $C$

   - there is an IND coming in $\sigma_i^+ : G_{i-1}[X_{i-1}] \subseteq G'_i[Y_i]$, and
   - there is an IND going out $\sigma_i^- : G'[i][X_i] \subseteq G'_{i+1}[Y_{i+1}]$, and
   $G'_j \in \text{descendants}\_or\_self(G_i)$

**Retaining Proper Circular Property**

Given the above requirements for retaining the acyclic property, we want to extend the property to proper circular INDs.

Recall, in Definition 2.10.2, that for any two consecutive INDs $I_i: R_{i-1}[X_{i-1}] \subseteq R_i[X_i']$, and $I_{i+1}: R_i[X_i] \subseteq R_{i+1}[X'_{i+1}]$ in a proper circular IND cycle, $X_i' = X_i$. Similarly, in a proper circular cycle of XML INDs, any two consecutive INDs $I_i: \tau_{i-1}[X_{i-1}] \subseteq \tau_i[X_i']$, and $I_{i+1}: \tau_i[X_i] \subseteq \tau_{i+1}[X'_{i+1}]$, $X_i' = X_i$. 

42
The transformation generated INDs are on newly created attributes 'id' and 'pid' that do not exist in the original sets of attributes. There is no chance the transformation generated INDs can form proper circular cycles with the original set of INDs. In other words, the transformation generated INDs may complete cycles, but they never complete proper circular cycles. As a result, if there are no proper circular cycles in the original set of XML INDs, it is not possible to have proper circular cycles after transformation. Therefore, retaining proper circular property if the original set of of INDs I has no proper circular cycles, is the same as retaining acyclic property in the last Section.

However, if there are proper circular cycles in the original set of INDs, we want to make sure the transformation generated INDs do not interfere with the existing cycles. To avoid such interference, any IND in the proper circular cycle must not be: $G_i[X_i] \subseteq G_j[X_j]$ where $G_i \in \text{descendant}(G_j)$.

### 3.2.2 Retaining Reduced Property

Analogous to the projection definition in Definition 2.10.5 for relational FDs, we define projection for XML FDs below.

The projection of a set of FDs $F$ over $G_i$ onto a set of attributes $Y \subseteq G_i$, denoted by $F[Y]$, is given by $F[Y] = \{ G_i : W \rightarrow Z | R_i : W \rightarrow Z \in F^+ \text{ and } WZ \subseteq Y \}$.

Again, analogous to the reduced definition in Definition 2.10.6 for relational constraints, we define the property reduced for XML constraints below.

**Definition 3.2.1** A set of attributes $Y \subseteq G_i$ is said to be reduced with respect to $G_i$ and a set of FDs $F$ over $G_i$ (or simply reduced with respect to $F$ if $G_i$ is understood from context) if $F[Y]$ contains only trivial FDs. A set of FDs and INDs $\Sigma = F \cup I$ is said to be reduced if $\forall G_i[X] \subseteq G_j[Y] \in I$, $Y$ is reduced with respect to $F$.

This definition is parallel to its relational counterpart. The transformation generated INDs involving attributes 'id' and 'pid' are all unary INDs. Thus they cannot violate the reduced property. Moreover, the transformation generated FDs only have 'id's on the L.H.S. of the FDs. Since none of the original INDs have attributes involving 'id's and the transformation generated INDs are all unary, the transformation generated FDs do not cause violation to a set of FDs + INDs that
are reduced. Therefore, the above XML reduced property definition is sufficient without the need to consider transformation generated constraints.

3.2.3 Inability to Retain Intersection Property

The transformation generated FDs are in the form: \( R_i: \text{'id'} \rightarrow A, \forall A \in R_i \). Thus it will violate the intersection property (Definition 2.10.4) if there are any non-trivial FDs in the original set of FDs.

The violation arises because \( \forall \) FD \( F = R_i: X \rightarrow A \) in the original set of FDs, \( A \neq \text{'id'} \) and \( A \neq \text{'pid'} \). On the other hand, a transformation generated FD exists \( R_i: \text{'id'} \rightarrow A \), and \( \text{'id'} \cap X = \emptyset \). Since we assumed \( F \) is not a trivial FD, it is not possible for \( \emptyset \rightarrow A \).

Therefore, it is impossible to retain intersection property if the original set of FDs is not empty.

3.2.4 Retaining Split-freeness Property

In the last section, we have shown the inability to retain intersection property. Luckily, it is not a necessary requirement. Non-interaction can still be achieved if split-freeness property (Definition 2.10.3) can be retained.

Two nontrivial XML FDs of the forms \( G: XB \rightarrow A \) and \( G: YA \rightarrow B \) are said to be cyclic.

**Definition 3.2.2** A set of FDs \( F \) over DTD \( \Delta_S \) satisfies the split-freeness property if \( \forall F_j \in F \); whenever there exist cyclic FDs, \( G_j: XB \rightarrow A, YA \rightarrow B \in F_j^+ \), then either \( G_j: Y \rightarrow B \in F_j^+ \) or \( G_j: (X \cap Y)A \rightarrow B \in F_j^+ \).

The above definition is parallel to its relational counterpart. The transformation generated FDs always have 'id' on the L.H.S., while 'id' is on the R.H.S. of FDs in neither the original set of FDs nor the transformation generated FDs. Thus they never form cyclic FDs with themselves or FDs from the original set of FDs, and split-freeness retention is ensured.
3.2.5 Summary of non-interaction between FDs and INDs

We have demonstrated the inability to retain the intersection property in Section 3.2.3, but defined properties of proper-circular in Section 3.2.1, split-freeness in Definition 3.2.2, and reduced in Definition 3.2.1 in XML context. A set of XML constraints satisfying these properties will have their transformed relational constraints satisfying proper-circular (Definition 2.10.2), split-freeness (Definition 2.10.3), and reduced (Definition 2.10.6) properties too. Therefore, the transformed constraints can enjoy non-interaction between FDs and INDs. This is a big step towards achieving efficient XML constraint implication.

3.3 Efficient implication of XML constraints

In Section 3.1, we have established that the problem of testing the implication of XML constraints can be reduced to the problem of testing the implication of relational constraints. We have also defined the necessary conditions in order for the transformed FDs and INDs not to interact. As a result, we can test implication of IND and FD independently.

In Section 3.3.1, we will show that typed XML INDs become typed relational INDs after the transformation in Section 3.1.4. After that, we describe a polynomial implicational algorithm for typed relational INDs in Section 3.3.2. At the end, we briefly outline a linear implication solution for relational FDs in Section 3.3.3.

3.3.1 Retaining Typed property

[1] shows relational FD implication has a linear solution, but IND implication is PSPACE-complete; therefore, we need an efficient solution. Fortunately, typed relational IND implication is polynomial. Using the typed XML IND defined in Definition 2.9.2, we show the typed property is retained after transformation.

It is easy to see that the original set of INDs do not violate the typed property after transformation. However, the transformation introduces extra INDs involving 'id' and 'pid'.

We define a new type for 'id' from each transformed relational table. Each 'pid' can only be involved in one IND, and its R.H.S. is an 'id' in another table. Thus this 'pid' belongs to the same type as the type of the target 'id', and these
transformation generated INDs do not cause typed violations among themselves. Since these new INDs and the original set of INDs deal with two mutually exclusive sets of attributes; in addition, FDs and INDs don’t interact, there is no interaction of the two sets of INDs.

Therefore, the typed property is retained after transformation.

3.3.2 Implication for typed relational INDs

[7] shows the implication of typed relational INDs can be solved in polynomial time by solving a digraph reachability problem. We will describe the solution below.

Given a set of relational schemas \( R = \{ R_1[U_1], \ldots, R_n[U_n] \} \), and a set of typed INDs \( \Sigma \) over \( R \). We want to check if \( \Sigma \) implies a typed IND \( \sigma = R_k[X] \subseteq R_l[X] \).

Construct a digraph \( G_X = (V, E) \), where \( V = \{ R_1, \ldots, R_n \} \) and \( (R_p, R_q) \in E \) iff there is \( R_p[W] \subseteq R_q[W] \) in \( \Sigma \) such that \( X \subseteq W \). \( \Sigma \models \sigma \) iff there is a path from \( R_k \) to \( R_l \).

Checking for existence of such a path is a digraph reachability problem, and it can be solved by breadth first search (BFS). Let the number of INDs be \( m \), and the total number of attributes (union of attributes from all relation schemas) be \( n \), the time complexity for BFS is \( O(m) \), and the space complexity is \( O(m) \). However, testing for attribute subsets during digraph building takes \( O(n^2) \) time, but \( O(n) \) space (time complexity can be lowered if we accept a trade-off for higher space complexity). Therefore, the overall time complexity is \( O(m+n^2) \), and the overall space complexity is \( O(m+n) \).

3.3.3 Implication for relational FDs

Relational FD implication is linear time [1]. [1] also describes the algorithm to test FD implication. The algorithm tests for implication by calculating the FD closure (attributes that the input set of attributes implies) on the L.H.S. attributes of the FD being tested.
3.3.4 Summary of XML Constraints Implication

In this chapter, we have reduced XML constraint implication to a relational constraint implication problem. Trying to find an efficient implication algorithm, we have adopted proper-circular (Definition 2.10.2), split-freeness (Definition 2.10.3), and reduced (Definition 2.10.6) properties from [11]. We modified them to apply to XML FDs and INDs. As a result, we have shown how to prevent interaction between XML FDs and INDs. In addition, to make the problem of IND implication tractable, we have introduced the typed property in XML. Finally, we have employed existing efficient implication algorithms on relational FDs and typed INDs to efficiently test implication of XML constraints.
Chapter 4

Constraint Propagation

When data is translated in the form of one schema to another, it carries some of the semantics with it. In this section, we introduce algorithms to translate constraints from a source DTD $\Delta_S$ to a target DTD $\Delta_T$ via a set of HepToX mapping rules $M$ (defined in Section 2.7).

As we will soon find out the constraint translation algorithms are decidable, but take exponential time because our algorithms compute closures. The exponential complexity makes this solution inefficient, especially when a large set of constraints are considered.

Computing a cover of FDs on the decomposed relations in relational functional dependency preservation also takes exponential time. Similar to the alternate method in dependency preservation, we define algorithms to check constraint propagation. That is given a set of constraint $\Sigma_S$ on $\Delta_S$, and a constraint $\sigma_T$ on $\Delta_T$, whether $\forall$ XML database $D_S$ on $\Delta_S$, if $D_S \models \Sigma_S$ implies $M(D_S) \models \sigma_T$.

4.1 Constraint Translation

Data can be translated via mappings; data semantics can also be translated. In this section, we construct algorithms to translate XML constraints via HepToX mappings.

Given a set of constraints $\Sigma_S$ on $\Delta_S$, $\delta(\Sigma_S, M)$ denotes the translation of $\Sigma_S$ via mapping rules $M$ to a set of target constraints. $\delta(\Sigma_S, M)$ produces a set of
constraints on the target DTD $\Delta_T$ such that:

$$\forall \text{XML database } D_S \text{ on } \Delta_S, \text{ if } D_S \models \Sigma_S, \text{ then } M(D_S) \models \delta(\Sigma_S, M)$$

If $\Sigma_S$ satisfies the conditions set in Section 3.2, there will be no interaction of FDs and INDs in constraint implication. Since we are translating source constraints, the non-interaction is very important to make the translation efficient because dependency preservation requires implication (possibly computation of closures). Therefore, we assume $\Sigma_S$ satisfies conditions in Section 3.2.

Our problem shares some similarities to the (functional) dependency preservation problem in the relational context. In fact, for the problem of FD translation, it employs the same algorithms as dependency preservation.

### 4.1.1 FD Translation

Each mapping rule $M_i \in M$ has a head and body. The rule head is a tree expression of a path from the target root to a group node. On the other hand, the rule body is a tree expression from the source root to multiple source group nodes.

Each target attribute is only allowed to have one correspondence. With this restriction, there does not exist the problem of union of values from different sources. Therefore, FDs translated from one source to a target group will not be interfered by data from other sources.

With the above established, we introduce the FD translation algorithm in Figure 4.1. As in dependency preservation, this is an algorithm that requires computation of FD closures; thus it requires exponential time. The algorithm is simple and helps to express the intuition behind FD translation.

Given a set of source FDs $F_S$, the algorithm first computes the closure $F_S^+$ of $F_S$. It then examines the leaf target group $TG_i$ of each mapping rule $M_i$ to see if any of the FDs in $F$ have all their attributes mapped to $TG_i$. If an FD satisfies these conditions, then it is translated to its form in $TG_i$ and added to the output set of target FDs $F_T$.

The actual FD translation at line 7 is very simple. Group instances of $SG_j$ will have their data in $X \cup Y$ transferred unchanged and as a whole to attributes group instances $U \cup V$ of $TG_i$. Therefore our translation is sound.
Algorithm $\delta(F_S,M)$

Input: a set of FDs $F_S$ over source DTD $\Delta_S$,  
a set of mapping rules $M$ from $\Delta_S$ to $\Delta_T$  
Output: a set of FDs $F_T$ over target DTD $\Delta_T$  
1. compute $F_S^+ = \{SG : X \rightarrow Y \mid F_S \models SG : X \rightarrow Y\}$  
2. for each $M_i \in M$  
3. $M_i$ has a rule head which contains a target path $TP_i$,  
   $TP_i$ is a path of groups root,...,$TG_i$  
4. for each group $SG_j \in$ rule body of $M_i$  
5. for each FD $=SG_j;X \rightarrow Y \in F_S^+$  
6. if $M_i$ maps $X \cup Y$ to attributes $U \cup V$ in $TG_i$,  
   where $|X| = |U|$ and $|Y| = |V|$  
7. then $F_T = F_T \cup \{TG_i : U \rightarrow V\}$

Figure 4.1: FD Translation Algorithm $\delta(F_S,M)$

This FD translation algorithm runs in exponential time; thus it is not very efficient.

4.1.2 IND Translation

IND Translation Algorithm as shown in Figure 4.2 translate a set of INDs $I_S$ on the source schema $\Delta_S$, to a set of INDs $I_T$ on the target schema $\Delta_T$.

Line 2,3 finds out all the DTD groups on both source and target schemas, and label each of them by indexes. Line 6-10 finds all the correspondences between DTD groups and store them in a 2-dimensional array $W$. We then compute $I_S^+$ closure of $I_S$ (all the INDs that can implied by $I_S$). If any IND $I' \in I_S^+$ has attributes on its L.H.S. fully mapped to a DTD target group $TG_{p'}$, and attributes in its R.H.S. fully mapped to another DTD target group $TG_{q'}$. Then we add this newly translated IND to $I_T$.

Even though typed IND implication is polynomial, $\delta(I_S,M)$ computes closure of $I_S$; thus it runs in exponential time and is not an efficient algorithm.
Algorithm \( \delta(I_S,M) \)

Input: a set of INDs \( I_S \) over source DTD \( Deltas \),
a set of mapping rules \( M \) from \( Deltas \) to \( DeltaT \)

Output: a set of INDs \( I_T \) over target DTD \( DeltaT \)

1. \( I_T = \{ \} \)
2. examine \( M \), find out all the source groups \( SGs \) and target groups \( TGs \)
3. index \( SGs \) and \( TGs \)
4. \( n = \text{sizeOf}(SGs) \)
5. \( m = \text{sizeOf}(TGs) \)
6. create a 2-dimensional array \( W \) of size \( n \times m \)
7. foreach \( M_i \in M \)
8. \( M_i \) has a rule head which contains a target path \( TP_i \),
    \( TP_i \) is a path of groups root, \ldots, \( TG_i \)
9. foreach group \( SG_j \in \text{rule body of } M_i \)
10. \( W[j][i] = \{(A_{S,p},A_{T,q}) | A_{S,p} \in R(SG_j), A_{T,q} \in R(TG_i), \) there is a correspondence between \( A_{S,p} \) and \( A_{T,q} \) in \( M_i \}\)
11. compute \( I_T^j \) from \( I_S \)
12. foreach \( I \in I_T^j \), where \( I = SG_p[X] \subseteq SG_q[Y] \)
    for \( r = 1 \) to \( m \)
13. if \( \exists (A,A') \in W[p][r], \forall A \in X \)
14. \hspace{1cm} p' = r
15. if \( \exists (B,B') \in W[q][r], \forall B \in Y \)
16. \hspace{1cm} q' = r
17. \( X' = \{ \} \)
18. \hspace{1cm} foreach A \in X
19. \hspace{2cm} X' = X' \cup \{A'\}, where \( (A,A') \in W[p][p'] \)
20. \( Y' = \{ \} \)
21. \hspace{1cm} foreach B \in Y
22. \hspace{2cm} Y' = Y' \cup \{B'\}, where \( (B,B') \in W[q][q'] \)
23. \( I_T = I_T \cup \{TG_{p'}[X'] \subseteq TG_{q'}[Y']\} \)

Figure 4.2: IND Translation Algorithm — \( \delta(I_S,M) \)
4.2 Testing Constraint Propagation

Given the undesirable exponential time complexity of constraint translation in Section 4.1, we want to find an efficient solution to check constraint propagation. In this section, we develop efficient algorithms to test constraint propagation. We give our problem statement below:

Given a set of constraints \( \Sigma_S \) on source DTD \( \Delta_S \), a set of mapping rules \( M \), and a constraint \( \sigma_T \) on target DTD \( \Delta_T \), \( \sigma_T \) is propagated from \( \Sigma_S \) via \( M \) iff \( \forall \) XML database \( D_S \) that conforms to \( S \) and \( \Sigma_S, M(D_S) \models \sigma_T \), and every attribute in \( \sigma_T \) has a correspondence in \( S \). We denote this \( \Sigma_S \models_M \sigma_T \).

Unlike traditional dependency preservation in which all the target attributes are mapped, our users may not create correspondences for all the attributes. Although a constraint may be satisfied at the target because of missing values (from target attributes without correspondences), we feel a constraint cannot be considered 'propagated' if some of its attributes are not mapped. Therefore, we require all attributes in \( \sigma_T \) to have a correspondence in \( \Delta_S \).

Even though we have already established polynomial implication of XML constraints, testing constraint propagation is still not a trivial problem. We have to be very careful to construct our environment to achieve decidability and tractability.

Firstly, users are allowed to create high-level correspondences between schemas that greatly reduce complications. The forbidding of multiple source correspondences to a single target node takes away the union operator. It guarantees data will come from only one part of the source.

Secondly, one may suspect that the set of translated constraints will not satisfy conditions set in Theorem 2.10.7; thus may lead to undecidability in testing propagation. In target constraint implication \( \Sigma_T \models \sigma_T \), all XML database \( D_T \) that conforms to \( \Delta_T \) and \( \Sigma_T \) has to satisfy \( \sigma_T \). However, in constraint propagation, we are only considering a subset of databases, namely \( \forall D_S \) that conforms to \( \Delta_S \) and \( \Sigma_S \), \( M(D_S) \models \Delta_T \), and we need to check if \( M(D_S) \models \sigma_T \). If we can test the constraint in question \( \sigma_T \) in its corresponding form in the source, then we can guarantee translated data coming from that part of the source will always satisfy \( \sigma_T \).

Thirdly, there is the issue of NULL and missing values. NULL values can occur if data coming from the source already contains NULL values, or an attribute in
the target has no correspondence (not mapped) in the source.

In the prior case, if the NULL values in a source XML database $D_S$ violates source constraint $\Sigma_S$, then $D_S$ will be violating the assumption of constraint propagation. Therefore, there is no propagation.

On the other hand, if $\forall D_S \models \Sigma_S$ and $D_S$ does not contain NULL values, $M(D_S) \models_M \sigma_T$, then $\forall D_S \models \Sigma_S$ and $D_S$ contains NULL values, $M(D_S) \models_M \sigma_T$. This is because when a $D_S$ with NULL values satisfies $\Sigma_S$, we can substitute unique values in place of these NULL values to produce $d'_S$, and it will still satisfy $\Sigma_S$. If $M(d'_S) \models \sigma_T$, then $M(D_S) \models \sigma_T$ because these unique values generated behave exactly as NULL values in this case as they do not equate to any other values.

Therefore, the existence of NULL values do not prevent $\sigma_T$ from propagating from $\Sigma_S$.

In the latter case, the target attribute has no correspondences in the source; thus it is impossible to find a corresponding source constraint. It follows if $\sigma_T$ being tested involves such attributes, it is not propagated.

As with the previous algorithms, we are able to guarantee no interaction of FDs and INDs in Section 3.2; therefore, we can separate the testing of propagation into two algorithms; one will test the propagation of FDs while the other is on INDs.

4.2.1 Testing FD Propagation

The intuition behind the FD Propagation Testing algorithm (Figure 4.3) is to trace data from the source that will make up the data involving the FD $\sigma \{TG_k : X \rightarrow A\}$ in question. For such data to conform to $\sigma_T$ after translation, it must conform to some constraints in the source. Hence, we must first find out what these constraints are, then test if the FDs $\Sigma_S$ given in the source can imply them.

Due to the one-to-one nature of our mappings, every mapped target attribute must be traced to one and only one attribute in the source. Thus checking if these traced attributes satisfy a corresponding source FD is sufficient to test propagation of $\sigma_T$.

To find this FD in the source, the first attribute we should look at is $A$ on the R.H.S. of $\sigma_T$. In line 8,9, we use the binding variable $A$ to trace the mapped attribute $A'$ in the source. In doing so, we also learn the source group $SG_k$ in
which $A'$ belongs. Since the definition of FDs confines an FD to attributes within a single group and our mapping rules do not consider joins among source groups, the corresponding FD we need to check must be in $SG_k$ where $A'$ belongs.

On a side note, we only need to check attributes in $SG_k$ because of no interaction of FDs and INDs. Therefore, we don’t need to worry about attributes in other groups having an effect on the FDs in $SG_k$. This also explains why the algorithm only accepts FDs for $\Sigma_S$. We can easily separate INDs and FDs from each other in a set of mixed constraints, and feed this algorithm with just the FDs.

Next in line 11-14, we look for all the attributes in $X$ that are mapped from $SG_k$, and put all their corresponding source attributes into $X'$. Notice that $|X'| \leq |X|$, $X$ is a superset of mapped attributes from $X'$. Due to the augmentation rule of FD implication, if a subset of $X$ implies $A$, then $X$ must imply $A$.

At the end, we check if $\Sigma$ implies $SG_k : X' \models A'$, returns true if it does, and false otherwise.

The algorithm checks FD implication at the end, while the rest of the algorithm runs in time $O(m)$, where $m$ is the number of correspondences. Thus the algorithm runs in linear time because testing implication of FDs runs in $O(n)$, where $n$ is the number of attributes.

For example, we can check if LibFD#1:

$UBCLibrary.person : name \rightarrow DOB$

in Table 1.2 is propagated from $\Sigma_{UBCBookstore}$ via the set of Mapping Rules M in Figure 1.4. At Line 8 of our algorithm, we find that the R.H.S. attribute DOB of LiBFD#1 appears in the mapping rule marked as ‘1’ in Figure 1.4, and it is assigned the variable $DOB$ in target DTD group person. At Line 9, we find $DOB$ is bound to the attribute DOB of the source DTD group person in the body of mapping rule ‘1’. At Line 12, name, which appears on the L.H.S. of LiBFD#1, is assigned to $name$. At Line 13, $name$ is bound to the attribute name of the same source DTD group person as DOB. At Line 15, we test if $\Sigma_{UBCBookstore} \models UBCBookStore.person : name \rightarrow DOB$. This is of course true because the FD is the same as BookFD#1 in $\Sigma_{UBCBookstore}$; thus our algorithm returns true.
Algorithm TestingFDPropagation

Input: a set of FDs $\Sigma_S$ over source DTD $\Delta_S$
an FD $\sigma_T = T G_k : X \rightarrow A$ over target DTD $\Delta_T$,
a set of mapping rules $M$ that maps $\Delta_S$ to $\Delta_T$
Output: true iff $\Sigma_S \models_{\sigma_T} \sigma_T$

1. $M_k = \text{null}$
2. for $M_i \in M$
3. head$(M_i)$ is a target path of groups $T P_i = \text{root}, \ldots, T G_i$
4. if $T G_i = T G_k$
5. then $M_k = M_i$
6. if $M_k = \text{null}$
7. then return false
8. in head$(M_k)$ the leaf group assignment has the form:
   $T G_k \rightarrow f_k(\ldots)$ [
   $\ldots$
   $A \rightarrow \$A,
   $\ldots$
   ]
   we denote it as head$_{T G_k}(M_k)$
9. in body$(M_k)$, there exists a source group (ignoring its descendant groups):
   $S G_k \rightarrow S S G_k$ [
   $\ldots$
   $A' \rightarrow \$A,$
   $\ldots$
   ]
   we denote it as body$_{S G_k}(M_k)$
10. $X' = \{}$
11. for each $B \in X$
12. there is $B \rightarrow \$B$ in head$_{T G_k}(M_k)$
13. if there exists $B' \rightarrow \$B$ in body$_{S G_k}(M_k)$
14. then $X' = X' \cup \{B'\}$
15. return $\Sigma_S \models X' \rightarrow A'$

Figure 4.3: Testing FD Propagation Algorithm
4.2.2 Testing IND Propagation

Testing IND propagation is similar to testing FD propagation in the previous section in that we trace attributes through mappings to identify the one corresponding IND in the source and check if a set of source INDs $\Sigma_S$ implies it.

Similarly, because of no interaction of FDs and INDs (Section 3.2), we don’t have to consider FDs in our algorithm. In addition, our mapping rules use Cartesian product of source groups to product instances of target groups; thus there is no equality of attributes among source groups. Therefore, for each left and right hand sides of the traced IND, we only need to consider attributes within only one source group for each side.

Our algorithm traces source attributes of both L.H.S. and R.H.S. of the IND in question $\sigma_T = TG_p[X_1, \ldots, X_n] \subseteq TG_q[Y_1, \ldots, Y_n]$. It makes sure the traced set of attributes $X'$ of the L.H.S. belongs to only one source group, and the traced set of attributes of the R.H.S. $Y'$ also belongs to only one source group.

Notice that both $X'$ and $Y'$ are lists because the L.H.S. and R.H.S. sets of attributes in an IND are sequences, and their order matters.

Moreover, unlike in FD propagation, IND implication does not have augmentation (augmenting attributes to attributes already in the constraint produces satisfying constraints). Hence, $|X'| = |X| = |Y'| = |Y|$, and every attribute in IND must have a corresponding attribute in the source.

Finally, if we can find $X'$ and $Y'$ which both contain attributes within their respective source groups $SG_p$ and $SG_q$, then $\Sigma_S \models \sigma_T$ iff $\Sigma_S \models TG_p[X'] \subseteq SG_q[Y']$.

The algorithm checks IND implication at the end, while the rest of the algorithm runs in time $O(m)$, where $m$ is the number of correspondences. Thus the algorithms runs in $O(n^2)$ time, where $n$ is the number of attributes, because testing implication of typed INDs runs in $O(n^2)$.

For example, we can check if LibIND#1:

$UBCLibrary.publication.call#.author[name,DOB] \subseteq UBCLibrary.person[name,DOB]$ in Table 1.2 is propagated from $\Sigma_{UBCBookstore}$ via the set of Mapping Rules $M$ in Figure 1.4. At Line 13 of our algorithm, we find that the L.H.S. attributes $name$ and $DOB$ of LiBIND#1 appear in the mapping rule marked as ‘2’ in Figure 1.4, and
Algorithm Testing IND Propagation

Input: a set of INDs $\Sigma_S$ over source DTD $\Delta_S$, an IND $\sigma_T = TG_p[X_1, \ldots, X_n] \subseteq TG_q[Y_1, \ldots, Y_n]$ over target DTD $\Delta_T$, a set of mapping rules $M$ that maps $\Delta_S$ to $\Delta_T$
Output: true iff $\Sigma_S \models \sigma_T$

1. $M_p = M_q = \text{null}$
2. for $M_i \in M$
3. head$(M_i)$ is a target path of groups $TP_i = \text{root}, \ldots, TG_i$
4. if $TG_i = TG_p$
5. then $M_p = M_i$
6. if $TG_i = TG_q$
7. then $M_q = M_i$
8. if $M_p = \text{null}$ or $M_q = \text{null}$
9. then return false
10. $SG_p = SG_q = \text{null}$
11. $X' = Y' = ()$
12. for each $X_i \in (X_1, \ldots, X_n)$
13. if $\not \exists X_i \rightarrow \$X_i \in \text{head}(M_p)$
14. then return false
15. there exists $X_i' \rightarrow \$X_i \in \text{a source group $SG \in body(M_p)$}$
16. if $SG_p \neq \text{null}$ and $SG_p \neq SG$
17. then return false
18. $SG_p = SG$
19. $X'.append(X'_i)$
20. for each $Y_i \in (Y_1, \ldots, Y_n)$
21. if $\not \exists Y_i \rightarrow \$Y_i \in \text{head}(M_q)$
22. then return false
23. there exists $Y_i' \rightarrow \$Y_i \in \text{a source group $SG \in body(M_q)$}$
24. if $SG_q \neq \text{null}$ and $SG_q \neq SG$
25. then return false
26. $SG_q = SG$
27. $Y'.append(Y'_i)$
28. return $\Sigma_S \models SG_p[X'] \subseteq SG_q[Y']$

Figure 4.4: Testing IND Propagation Algorithm
they are assigned the variables $\text{name}$ and $\text{DOB}$ in target DTD group \textit{publication} respectively. At Line 15, we find $\text{name}$ and $\text{DOB}$ are bound to the attributes \textit{author:name} and \textit{author:DOB} of the source DTD group \textit{book} in the body of mapping rule ‘1’. At Line 21, we find that the R.H.S. attributes \textit{name} and \textit{DOB} of LiBIND#1 appear in the mapping rule marked as ‘1’ in Figure 1.4, and they are assigned the variables $\text{name}$ and $\text{DOB}$ in target DTD group \textit{person} respectively. At Line 23, we find $\text{name}$ and $\text{DOB}$ are bound to the attributes \textit{name} and \textit{DOB} of the source DTD group \textit{person} in the body of mapping rule ‘1’.

At Line 28, we test if:
\[
\Sigma_{\text{UBCBookstore}} \models UBCBookStore.\text{subject.book}[\text{author.name,author.DOB}] \subseteq UBCBookStore.\text{person}[\text{name,DOB}]
\]
. This is of course true because the IND is the same as BookIND#1 in $\Sigma_{\text{UBCBookStore}}$; thus our algorithm returns true.

4.3 Constraint Propagation Summary

To test constraint propagation, we first attempted to translate all source constraints to a cover of target constraints, then one could use implication algorithms in Chapter 3 to check if the target constraint $\sigma_T$ is implied by this cover. However, testing constraint involves the computation of closures; thus making the algorithm to run in exponential time. Therefore, we developed constraint propagation testing algorithms that avoids computing a cover of target constraints by translating $\sigma_T$ back to its form in source schema, and testing if $\Sigma_S$ implies it. The propagation testing algorithms run in polynomial time.
Chapter 5

Detecting Constraint Violations

When a constraint propagates, it guarantees that translated data does not violate the constraint at the target. However, when one does not propagate, it does not necessarily imply violation.

For example, in Figure 1.1 it is obvious that LibFD#3 in the target DTD UBCLibrary is propagated; thus it will be satisfied by data translated from UBCBookstore using the specified mapping.

However, LibFD#7 is not propagated. According to the mapping rules, no data will be translated under the group publisher. Since translated instances never contain publisher, LibFD#7 cannot be violated.

This example demonstrates that propagated constraint implies satisfaction, but constraint satisfaction does not necessarily imply propagation.

As in checking constraint propagation, we can check for the violation of a target FD by only a set of source FDs and mapping rules independent of any INDs, and vice versa for checking IND violation.

In the following two sections, we are going to introduce algorithms to check for FD and IND violations respectively.

5.1 Checking for FD violation

The algorithm Figure 5.1 presented below takes a set of FDs $\Sigma_5$ over source DTD $\Delta_5$, an FD $\sigma_T = TG_k : X \rightarrow A$ over target DTD $\Delta_T$, and a set of mapping rules $M$. It
Algorithm Checking for FD violation

Input: a set of FDs $\Sigma_S$ over source DTD $\Delta_S$,
an FD $\sigma_T = TG_k : X \rightarrow A$ over target DTD $\Delta_T$,
a set of mapping rules $M$ that maps $\Delta_S$ to $\Delta_T$
Output: true iff $\exists$ an XML instance $D_S$ over $\Delta_S$
that conforms to $\Sigma_S$, s.t. $M(D_S) \not\models \sigma_T$

1. if TestingFDPropagation($\Sigma_S$, $\sigma_T$, $M$)
2. then return false
3. $M_k = null$
4. for $M_i \in M$
5. head($M_i$) is a target path of groups $TP_1 = root, \ldots, TG_l$
6. if $TG_i = TG_k$
7. then $M_k = M_i$
8. if $M_k = null$
9. then return false
10. for each $X_i \in X$
11. if $X_i \rightarrow X_i \in head(M_k)$
12. then return false
13. return true

Figure 5.1: FD violation Checking Algorithm

determines if any data instances $D_S$ over $\Delta_S$ that conforms to $\Sigma_S$ would violate $\sigma_T$
after the translation of $D_S$ through $M$.

Propagation implies no violation; therefore, line 1,2 of the algorithm in Figure
5.1 check for propagation, and return false (no violation) if $\sigma_T$ is propagated from
$\Sigma_S$ through $M$.

For $M(D_S)$ to violate $\sigma_T$, we need all attributes on the L.H.S. of $\sigma_T$ to be equal
on two instances $x,y$ over group $TG_k$, but $x.A \neq y.A$. As our stance on NULL
and missing values is that they are not comparable; thus $x.X$ and $y.Y$ cannot be equal if
any of them contain NULL values or are missing some values. Hence, in order
to create violations, all attributes in $X$ must be mapped.

On the other hand, if all attributes in $X$ are mapped, whether $A$ is mapped is
irrelevant. If A is mapped, together with the fact that $\sigma_T$ is not propagated, we can always find $D_S$ such that there are translated instances $x,y$ in $M(D_S)$ and $x.X = y.X$ but $x.A \neq y.A$. If A is not mapped, then we just need to find $D_S$ such that there are translated instances $x,y$ in $M(D_S)$ and $x.X = y.X$. The absence of mapping of A guarantees $x.A \neq y.A$.

This algorithm first checks for propagation which runs in $O(n)$ time, where $n$ is the number of attributes. The rest of it runs in time $O(m)$, where $m$ is the number of correspondences; thus, we can check for FD violations in $O(n)$ time.

For example, we can check if LibFD#7:

```
UBCLibrary.publisher: name → addr
```

will cause violations after translation. It is obvious that LibFD#7 is not propagated; therefore, testFDPropagation at Line 1 is false. The algorithm go through all the mapping rules $M$ (Figure 1.4) at Lines 4-6, but will not find a mapping rule mapping to the publisher DTD target group node because there are not correspondences to publisher. Therefore, at Line 9, our algorithm returns false; thus it correctly answers LibFD#7 will not cause violations.

### 5.2 Checking for IND violation

The algorithm in Figure 5.2 takes a set of INDs $\Sigma_S$ over source DTD $\Delta_S$, an IND $\sigma_T = TG_p[X_1, \ldots, X_n] \subseteq TG_q[Y_1, \ldots, Y_n]$ over target DTD $\Delta_T$, and a set of mapping rules $M$. It determines if any XML database $D_S$ over $S$ that conforms to $\Sigma_S$ will violate $\sigma_T$ after the translation of $D_S$ through $M$.

Similar to Figure 5.1, we check for propagation first. If $\sigma_T$ propagates, then there cannot exist violations.

In order for $M(D_S)$ to violate $\sigma_T$ when $\sigma_T$ is not propagated, it must contain an instance $x$ of group $TG_p$, and $y$ of group $TG_q$, such that $\exists i \in [1, \ldots, n], x.X_i \neq y.Y_i$.

In this case the existence of mappings for any $X_i$ or $Y_i$ is irrelevant. This is because, if mappings are missing for any of $X_i$ or $Y_i$, the missing values guarantee non-equality; thus they lead to violation. On the other hand if all $X_i$ and $Y_i$ are mapped, we can easily find an XML instance $D_S$ over $\Delta_S$, with $D_S \models \Sigma_S$, s.t. $M(D_S) \not\models \sigma$. Failure to find such $D_S$ implies $\Sigma_S \models_M \sigma$ which contradicts our assumption that $\sigma_T$ is not propagated.
Algorithm Checking for IND violation

Input: a set of INDs $\Sigma_S$ over source DTD $\Delta_S$, an IND $\sigma_T = TG_p[X_1, \ldots, X_n] \subseteq TG_q[Y_1, \ldots, Y_n]$ over target DTD $\Delta_T$, a set of mapping rules $M$ that maps $\Delta_S$ to $\Delta_T$

Output: true if $\exists$ an XML instance $D_T$ over $\Delta_S$ that conforms to $\Sigma_S$, s.t. $M(D_S) \neq \sigma_T$

1. LHGroupFound=false
2. for $M_i \in M$
3. head($M_i$) is a target path of groups $TP_i$=root, ..., $TG_i$
4. if $TG_p \in TP_i$
5. then LHGroupFound=true
6. if LHGroupFound=false
7. then return false
8. if TestingINDPropagation($\Sigma_S, \sigma_T, M$)
9. then return false
10. else return true

Figure 5.2: IND violation Checking Algorithm

Therefore, the checking for IND violation is surprisingly simple; if any attributes in $TG_p$ or its descendants are mapped, then it is just the opposite outcome of testing IND propagation. Otherwise, since there can be no translated instances of $TG_p$, thus $\sigma_T$ is satisfied because there are no instances on the L.H.S. of $\sigma_T$ to violate it.

Checking existence of mappings in $TG_p$ or its descendants takes $O(m)$, where $m$ is the number of correspondences. Hence, the algorithm runs in $O(n^2)$ time, where $n$ is the number of attributes, because the IND propagation testing algorithm that we employ runs in $O(n^2)$.

For example, we can check if LibIND#2:

```
UBCLibrary.publication[pub] \subset UBCLibrary.publisher[name]
```

will cause violations after translation. At Line 3, the algorithm finds the L.H.S. target group "publication" in mapping rule marked as '2' in Figure 1.4. Since there will be instances on the L.H.S. of the IND, the algorithm employs the TestingIND-
Propagation algorithm in Section 5.1 to test if LibIND#2 propagates. The R.H.S. of LibIND#2 is not mapped; therefore, LibIND#2 does not propagate, and our algorithm returns true (meaning LibIND#2 will cause violations).
Chapter 6

Resolving Violations

With the ability to detect constraint violations, we are setup to fix violations in this chapter. However, violations may be caused by many factors such as human error and heterogeneous schemas. There is no way to determine the reasons behind the violations automatically; only the person who creates the correspondences can decide how best to rectify the violations.

The purpose of this chapter is to give options to the person trying to repair constraint violations. Each solution has its advantages and disadvantages, and we will discuss them in the following sections. Which method is best suitable for a correction differs depending on different situations. Therefore, we will leave the final decision making to administrators creating the correspondences.

Unlike the rest of this thesis, the repairs presented here are heuristic solutions. There is no guarantee that one repair won’t cause other violations. They merely provide options for administrators to produce better quality mappings.

We categorize the repairs into repairs that fix FD violations and the ones that fix IND violations.
6.1 Repairing FD Violations

6.1.1 Scaling Back

Studying the algorithms in Detecting Constraint Violations Chapter 5, one can easily come to the conclusion that removing some of the correspondences can fix violations. This is indeed a method that is consider the safety net because it works in all situations. However, this has to be considered a pessimistic option as the resulting mappings will translate less information after deleting some correspondences.

In our example in Figure 1.1, we have shown that the FDs LibFD#4 and LibFD#5 are violated. Here, in order to demonstrate better, we simplify our example by retaining only attributes and correspondences that are necessary for our demonstration, and it is shown in Figure 6.1.

An easy way to fix the violations is to delete one of the correspondences to subject, author.name, or year. By doing so, there will be missing values of the L.H.S. of LibFD#4 and LibFD#5. Hence, L.H.S. of LibFD#4 and LibFD#5 will never equate, and the two FDs cannot be violated.

Formally, if FD \( \sigma = TG_k : SP_1, \ldots, SP_n \rightarrow SP_u \) is violated, then delete the correspondence for \( TG_k.SP_i \) for any \( i \in [1, \ldots, n] \)

In our example, if we delete the correspondence between years of the two
schema, the translated data will be missing the attribute year. The resulting data will satisfy LibFD#4 and LibFD#5 because of the missing values of year.

6.1.2 Source Concatenation

Scaling back deletes correspondences; thus some information is lost. It can be undesirable. As seen in the last section if we delete the correspondence between years of the two schemas in Figure 6.1 the resulting data will satisfy LibFD#4 and LibFD#5. However, the resulting translated database will be missing year; users are going to have a hard time searching for publications.

In this section, we introduce a method that let us keep as much information as possible while satisfying target FDs. Here, we introduce a new type of correspondence, the concatenating correspondence.

A concatenating correspondence connects multiple source attributes within one DTD group to a single target attribute. During translation, each source group instance will concatenate the strings of the source attributes of the correspondence and put the resulting string in the target attribute.

![Diagram](image)

**Figure 6.2: Fixing FD Violation by Concatenation Example**

Figure 6.2 shows a concatenating correspondence connecting isbn and year at the source to year at the target. Translating the instance from Figure 6.2 will produce Figure 6.3. It is clear that the new translated instances no longer violate
LibFD#4 and LibFD#5. This is achieved by concatenating attributes that implies the corresponding R.H.S. of target FDs to attributes of L.H.S. of those FDs. In our example, isbn implies title and #pages which are correspondences of R.H.S. of LibFD#5 and LibFD#5. Thus we concatenate isbn to a correspondence that is in the L.H.S. of LibFD#4 and LibFD#5. Administrators establishing the correspondences need to pick attributes in the source that implies A', where A' is the source correspondence of A in the violated target FD $\sigma_T = TG_K : X \rightarrow A$. Which correspondence to concatenate to in X is totally up to the administrator establishing the correspondences. Here we choose the correspondence between year, and concatenate values in isbn to every year translated to UBCLibrary.

### 6.2 Repairing IND Violations

#### 6.2.1 Scaling Back

When an IND is violated, there are instance on the L.H.S. of the violated IND $\sigma_T$ that do not have corresponding instances on the R.H.S.. One way to avoid IND violation is to make sure no instances on the L.H.S. exist.

Again, we can achieve this by deleting correspondences such that no instances appear on the L.H.S. of $\sigma_T$. Since we treat missing values in a group instance as NULL values, simply deleting the correspondences that makes up the L.H.S. attributes is not enough. We need to delete all the correspondences related to the
target group so that no group instances will ever appear.

Figure 6.4 shows just the part of our running example that violates LibIND#2. Following the correspondences, instances of publications are translated, each publication has a pub. LibIND#2 requires each pub in publication to appear in publisher.name, but this is not satisfied by our correspondences.

Using the correspondence deleting method, we have to delete every correspondence in the publication. There will not be any more publication instances; thus LibIND#2 will no longer be violated.

As with the Scaling Back solution of FD violation repairs, deleting correspondences loses information. It can be considered worse in this case because all correspondences of the L.H.S. target group have to be deleted.

6.2.2 Split Correspondence

Instead of making sure the L.H.S. of an IND $\sigma_T = TG_k[SP_{k_1}, \ldots, SP_{k_n}] \subseteq TG_l[SP_{l_1}, \ldots, SP_{l_n}]$ has no instances, we can make sure the R.H.S. contains all the instances of the L.H.S.

A split correspondence is actually multiple correspondences all originating from the same source attribute. After translation, all target attributes will have the same set of source attributes.

This is a constructive method to correct IND violations. For every mapped attribute $TG_k.SP_{k,l}$ on the L.H.S. of $\sigma_T$, we find its corresponding source attribute
Figure 6.5: IND Violation Example Fixed by Split Correspondence

$SG_k.SP_{k,i}$, and create another correspondence from $SG_k.SP_{k,i}$ to the corresponding attribute $TG_l.SP_{l,i}$ in the R.H.S. of the IND.

For every unmapped attribute $TG_k.SP_{k,i}$ on the L.H.S. of $\sigma_T$, we need to find any attributes in $SG_k$ to map to $TG_k.SP_{k,i}$ and $TG_l.SP_{l,i}$.

In Figure 6.4, our example violates LibIND#2. Using the split correspondence method, we create another correspondence to $publisher:name$ as shown in Figure 6.5. It is obvious that every translated $pub$ appearing under $publication$ will appear in $publisher.name$. Therefore, LibIND#2 is satisfied.

6.3 Summary of Repairs

In this chapter, we have suggested 3 heuristic ways to repair violations. Roll back of correspondences can repair all violations, but it can be undesirable due to loss of information. Concatenation and split correspondences are more constructive, but they may modify translated data. Therefore, administrators have to decide which type of repair to apply, and if the sacrifice is worth the gain in constraint satisfaction.
Chapter 7

Conclusion and Future Work

Conclusion
We have proposed ways of repairing correspondences such that translated instances would satisfy target constraints. In order to repair, we first have to know what we are repairing. Thus we have developed algorithms to detect violations, check propagation of constraints, and check implication of constraints.

To make sure our problem is tractable, we have investigated simplified DTD and restricted conditions for constraints such that the above algorithms all run in polynomial time. These efficient results are all on the two most fundamental type of constraints: functional dependencies and inclusion dependencies.

Our mappings are generated from high-level correspondences that can be easily created by anyone who knows the two schemas well. We suggest repairs on these high-level correspondences. In addition, repairing these high-level correspondences is easier for everyone to understand and to choose the best method of repairing.

With consistent mappings translating data that is consistent with the target constraints, application at the target are guaranteed to run soundly.

Future Work
This thesis opens up several new problems.

Data exchange can be more powerful if data from one source can be relayed to
other targets through chains of mappings. Instead of creating mappings between every pair of schemas we want data exchange, we can make use of existing mappings to form a network. Ideally, data exchange is possible if two schemas are connected via a path of mappings.

Our constraint propagation algorithm will be more powerful if we can test for constraints propagation along a chain of mappings.

To reason on the most basics of constraints: functional dependencies and inclusion dependencies, we have yet to study some of the other popular constraints such as keys and foreign keys. Considering keys and foreign keys involves more studies of non-leaf nodes, and treating groups as entities. This is quite different from our study of only leaf nodes. We only need to deal with sets of values. However, we believe our studies is a very useful building block towards keys and foreign keys violation detections and repairs.

Similar to the idea of relative keys in [6], there can be relative FDs and INDs. For a relative FD, the L.H.S. of it can involve ancestor attributes of the DTD group that the FD is based on. Relative IND is similar in that sequences of attributes on either sides of it can span over several DTD groups that are descendants and ancestors of each other.

However, with attributes spanning over multiple DTD groups, our constraint implication that based on reduction to relational constraint implication will no longer work. Therefore, we need to find another implication system that is efficient.
Bibliography


doi:{http://doi.acm.org/10.1145/303976.303983}. → pages 10, 16

doi:{http://doi.acm.org/10.1145/303976.303983}. → pages 5, 8

doi:http://doi.acm.org/10.1145/1066157.1066175. → pages 5, 8


